Edge Cloud -automaatio workflow-työkaluilla

Nokia Solutions and Networks Oy

Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäki, Tieto -ja viestintätekniikka

Kevät 2020

Teemu Kilpeläinen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto -ja viestintätekniikka
Riihimäki

Tekijä Teemu Kilpeläinen **Vuosi** 2020

Työn nimi Edge Cloud -automaatio workflow-työkaluilla

Työn ohjaajat Teemu Järvenpää - HAMK, Pasi Juvonen - Nokia

TIIVISTELMÄ

Tämän opinnäytetyön tavoitteena oli vertailla ns. workflow-työkaluja Edge Cloud -automaatiota varten. Työssä käytiin läpi eri työkalujen ominaisuuksia sekä tutkittiin hieman työkalujen sisäistä toimintaa. Työtä varten selvitettiin mitä ominaisuuksia työkalujen kuuluu täyttää kyseistä automaatiotarkoitusta varten.

Osana työtä oli pienimuotoisen demo-esityksen toteutus, jonka avulla selvitettiin onko haluttu automaatio mahdollista toteuttaa käyttäen erilaisia workflow-työkaluja. Opinnäytetyön performanssitesteissä käytettyjen workflow'ien luominen oli myös osa työtä.

Työssä käsiteltiin säiliötekniikoita ja selvitettiin hieman tarkemmin näiden tekniikoiden toimintaa. Työssä oli vertailussa erilaisia workflow-työkaluja ja perusteellinen analyysi tehtiin vertailu- ja performanssidatan sekä työssä työkaluille laadittujen vaatimusten perusteella.

Ominaisuusvertailussa selvisi, että työkaluista Argo-työkalu täytti suurimman osan vaatimuksista. Toisella sijalla oli Tekton-työkalu, joka päätyi toiselle sijalle muutamien puutteiden takia, vaikka työkalu oli toiminnaltaan hyvin samankaltainen Argon kanssa. Airflow jäi ominaisuusvertailussa viimeiselle sijalle, koska työkalu oli liian vanhanaikainen verrattuna muihin työkaluihin.

Performanssivertailussa Airflow oli nopein, Argo toiseksi nopein ja Tekton hitain. Airflow'n huomattava ero nopeudessa oli kiinni ympäristöstä, koska testeissä käytetty ympäristö oli yhden noodin klusteri, joten performanssi oli suuria klustereita varten suunnattujen Argo ja Tekton -työkalujen kohdalla huonompi.

Avainsanat Cloud, Kubernetes, Nokia, Workflow

Sivut 49 sivua, joista liitteitä 9 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Riihimäki

Author Teemu Kilpeläinen **Year** 2020

Subject Edge Cloud automation with workflow-tools

Supervisors Teemu Järvenpää - HAMK, Pasi Juvonen - Nokia

ABSTRACT

The goal of this thesis was to find out which workflow tool suits best for Edge Cloud automation. Analysis of features and functionality of different workflow tools was made and the features that Edge Cloud automation requires were evaluated.

A part of this thesis was to create a simple demo using these workflow tools to figure out if the tools are sufficient enough for this automation purpose. Basic performance testing was done for these tools and the creation of these test workflows was also a part of this thesis.

Container technologies were researched upon in this thesis and further analysis was done on the functionality of these technologies. The thesis had different workflow tools in comparison and analysis was made based on the evaluation and performance data.

Feature comparison showed that Argo had most of the features in our list of requirements. Tekton fell to the second place because of a couple of missing features, although the tool was overall close to the features of Argo. Airflow did not fit the automation purposes as well as the other tools did, because the tool was too old school compared with the other tools.

In performance comparison Airflow was the fastest tool, Argo second and Tekton last. Airflow's fast speed was related to the slow test environment, the environment was a one node -cluster and this slowed the performance of Argo and Tekton -tools, which are made for larger clusters.

Keywords Cloud, Kubernetes, Nokia, Workflow

Pages 49 pages including appendices 9 pages

SISÄLLYS

Liitteet

Liite 1 Argo testi Workflow

Liite 2 Tekton testi Pipeline ja muut resurssit

Liite 3 Airflow testi DAG

TERMISTÖ

Artifact repository	Argossa käytettävä objektisäiliö
Artificial Intelligence	Tekoäly
Bash	Komentotulkki Unix ja Linux ympäristöissä
CaaS	Container as a Service, Säiliö palveluna
CI/CD	Continuous Integration/Delivery
CLI	Command-line interface
Cloud-RAN	Cloud-Radio Access Networks
ConfigMap	Kubernetes deploymentin resurssitiedosto
Container	Säiliö
Control Plane	Podeja ja nodeja ohjaava järjestelmä
CRD	CustomResourceDefinition
CU	Control Unit
cURL	Työkalu datansiirtoa varten
Cron-scheduler	Unix pohjainen ajastuspalvelu sovelluksille
DAG	Directed acyclic graph
Data center	Datakeskus – rakennus jossa useita palvelimia
Docker image	Docker säiliöinstanssin levykuva
DU	Distributed Unit
End-to-end	Sovellus tai palvelu verkon loppupäässä
Gerrit	Git-työkaluun perustuva projektinhallinta työkalu.
Git	Versionhallinta työkalu
GSM tai 2G	Toisen sukupolven matkapuhelinstandardi
HA	High-Availability, korkea saatavuus
HTML	Hypertext Markup Language
Image	Levykuva
IoT	Internet of Things, Esineiden internet
Key-value tietokanta	Tietokanta, jossa data on linkitetty avaimiin
Kubernetes	Säiliöiden orkestraatiojärjestelmä
Kubernetes Deployment	Kubernetesilla suoritettu ryhmä, joka sisältää monia podeja.
Kubernetes klusteri	Usean palvelimen node-ryhmä
Kubernetes natiivi	Ohjelmisto, joka on rakennettu Kubernetes komponenttien avulla
Kubernetes node	Yksittäinen palvelin Kubernetes klusterissa
Linux	Käyttöjärjestelmä
Linux-kernel	Linux-käyttöjärjestelmän ydin
Machine Learning	Koneoppiminen
Master node	Kubernetes klusteria ohjaava palvelin
NMT-900	Ensimmäisen sukupolven matkapuhelin-standardi
NFVI	Network Functions Virtualization Infrastructure
Node	Kubernetes klusterin palvelin
Objektisäilö	Tietokanta, jossa data on objektimuodossa
OpenStack	Avoimen standardin pilvipalvelu
Pipeline	Putki tai jono, jossa tehdään yleensä CI/CD töitä

Pod	Kubernetesin säiliöitä ohjaava ryhmä
PVC	Persistent volume claim
Rack mount	Kehikkoon kiinnitettävä laite
Rack unit	Palvelin kehikon yksikkö
REST API	Verkkoprotokollan ohjelmointirajapinta
RU	Radio Unit
Virtuaalikone	Fyysistä tietokonetta mallintava virtuaalitietokone
Virtualisointi	Fyysisen laitteen abstraktio ohjelmistolla
Workflow	Työnkulku
Workflow template	Workflow mallipohja
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language
3G	Kolmannen sukupolven matkapuhelinstandardi
4G	Neljännen sukupolven matkapuhelinstandardi
5G	Viidennen sukupolven matkapuhelinstandardi

1 JOHDANTO

Opinnäytetyön selvityksen kohteena ovat workflow-työkalut. Työn tarkoituksena on selvittää ja tehdä vertailua Nokian Edge Cloud -automaatiota varten valituista workflow-työkaluista. Yrityksellä ei ollut kirjoituksen aikana Edge Cloud -tuotteen sisällä automaatiota. Edge Cloud -infrastruktuurin automaatiolla saavutettaisiin paljon helpompi laitteiston asentaminen ja konfigurointi verrattaessa manuaaliseen konfigurointiin. Opinnäytetyön kirjoituksen hetkellä asentaminen vaati vähintään viikon, että sai infran toimimaan, joten automaatio helpottaisi huomattavasti asennuksen prosessia.

Valitsin vertailuun työkaluiksi Argon, Tektonin ja Airflow'n, koska nämä olivat opinnäytetyön kirjoituksen hetkellä suosituimpia workflow-työkaluja. Työkalujen tutkimisessa hyödynsin projektien GitHub-projektisivuja ja muita dokumentaatioita. Itse vertailussa hyödynnettiin tätä tutkittua tietoa sekä työkalujen käytön yhteydessä löydettyjä vikoja/ominaisuuksia. Opinnäytetyön loppuun tehtiin yhteenveto, josta selviää miten projekti eteni, miten opinnäytetyö onnistui ja minkä työkalun valitsisin itse tähän käyttötarkoitukseen, sekä listattiin lyhyesti parannus- ja jatkokehitysehdotuksia.

2 NOKIA SOLUTIONS AND NETWORKS OY

Opinnäytetyön tilaaja oli Nokia Solutions and Networks Oy, joka toimii Espoossa Karamalmissa. Nokia Solutions and Networks Oy on osa Nokia Oyj:n liiketoimintayksikköä ja on toiminut ympäri maailmaa n. 150 maassa vuodesta 2007 lähtien. Yritys myy verkkolaitteita ja palveluita asiakkaille globaalisti. (Nokia, n.d.a.)

Yrityksen verkkoliikennetoiminta koostui opinnäytetyön kirjoituksen aikana kuudesta liiketoimintaryhmästä/yksiköstä. Nämä yksiköt olivat nimiltään Mobile Networks, Fixed Networks, IP/Optical Networks, Nokia Software, Global Services ja Nokia Enterprise. Myös monista Nobel-palkinnoista tunnettu tutkimus- ja kehitysyksikkö Bell Labs (Nokia, n.d.g.) siirtyi Nokialle vuonna 2015.

Nokialla on pitkä historia tietoliikennealalla ja yritys on tuottanut monenlaisia tuotteita yritys- ja yksityiskäyttöön. Yrityksen tuotteista tunnetuimpia ovat Nokian matkapuhelimet, mutta yritys on irtaantunut näistä markkinoista. Nykyisistä Nokia brändin puhelimista vastaa HMD Global. Opinnäytetyön kirjoituksen hetkellä Nokia kehitti ja myi pääosin verkkoinfrastruktuuriin ja verkkolaitteistoon liittyviä tuotteita.

2.1 Yrityksen historiaa

Nokia alkoi toimia radioverkkojen parissa 1970-luvun alussa ja alkuun suurin osa radiolaitteistosta myytiin Suomen puolustusvoimien käyttöön. 1980-luvulla Nokia oli osana kehittämässä GSM-mobiilistandardia matkapuhelimille ja Siemensin kanssa yhteistyönä rakensivat ensimmäisen toimivan GSM-verkon vuonna 1991.

Nokia-Mobiran (nyk. Nokia) ensimmäinen täysin langaton matkapuhelin oli Mobira Cityman 900, joka tuli markkinoille vuonna 1987. Tämä puhelin käytti NMT-900 (Wikipedia, 2020) mobiilistandardia. Nokian ensimmäinen GSM puhelin Nokia 1011 (Kuva 1) saapui markkinoille vuonna 1992.

Kuva 1. Nokia 1011 GSM-puhelin, (Mobile phone history, n.d)

Vuonna 1998 Nokia ohitti Motorolan ja vei maailman myydyimmän matkapuhelinbrändin paikan. Tästä seuraava iso askel matkapuhelimissa oli Nokia 6630, joka oli yrityksen ensimmäinen 3G-puhelin, joka julkaistiin vuonna 2004. Tästä lähtien Nokia oli suurimmillaan ja vuonna 2007 saavutti yrityksen historian parhaat myyntitulokset.

Myynnin kanssa alkoi olla ongelmia, kun markkinoille tuli kilpailevia vaihtoehtoja esimerkiksi vuonna 2007 julkaistu Applen iPhone. Nokian puhelimien Symbian-käyttöjärjestelmä ei ollut enää kilpailukykyinen ja myynti alkoi putoamaan.

Nokia oli kehittänyt Maemo-laitteita jo pitkään ja perusti MeeGo-projektin yhteistyönä Intelin kanssa. MeeGo (Wikipedia, 2020b) oli Linux käyttöjärjestelmä mobiililaitteille, jonka oli tarkoitus kilpailla muiden mobiilikäyttöjärjestelmien kanssa. Tästä huolimatta vain yksi puhelin - Nokia N9 (Kuva 2) julkaistiin tämän käyttöjärjestelmän ”Harmattan”-versiolla (Wikipedia, 2020c). Vaikka puhelin otettiin positiivisesti vastaan, päätti yritys tästä huolimatta aloittaa Windows Phone käyttöjärjestelmällä varustettujen puhelimien kehityksen.

Kuva 2. Nokia N9 matkapuhelin, (The Verge, 2020)

Alkaen vuodesta 2011 Nokia teki Microsoftin kanssa yhteistyönä Lumia Windows Phone -matkapuhelimia, jotka eivät saavuttaneet samanlaista suosiota kuin aiemmat puhelimet. Vaikka käyttöjärjestelmä oli nopea ja täynnä ominaisuuksia, oli ongelmana liian myöhäinen saapuminen markkinoille. Android ja iOS olivat olleet jo pitkään markkinoilla. Lumia-puhelimien sovelluskauppa oli suhteellisen tyhjä alusta loppuun saakka, koska alustalle ei yksinkertaisesti ollut tarpeeksi kolmannen osapuolen kehittäjiä.

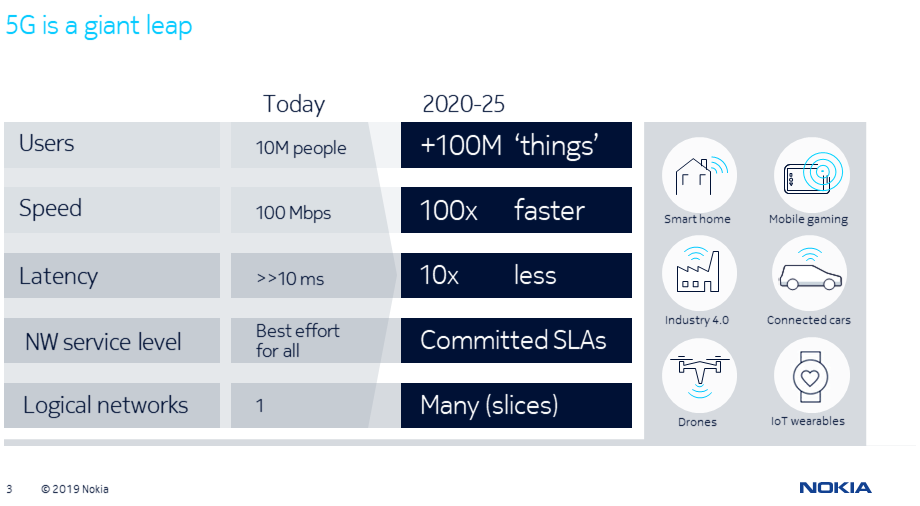
Vuonna 2013 Nokia ilmoitti myyvänsä mobiilitoimintansa Microsoftille, eikä ei enää tämän jälkeen ole tehnyt Nokia-brändin puhelimia. Vuonna 2016 Microsoft myi Nokialta ostetut mobiiliyksiköt HMD Global -nimiselle yritykselle, joka valmisti vielä opinnäytetyön kirjoituksen hetkellä uusia Nokia-brändin puhelimia markkinoille.

2.2 5G

Opinnäytetyön kirjoituksen aikana Nokian myynnissä ja kehitteillä olevista tuotteista suurin oli 5G. 5G on viidennen sukupolven mobiiliverkko-teknologia, jota käytetään datan siirtoa varten esimerkiksi älypuhelimissa ja muissa verkkoon kytketyissä laitteissa. 5G on iso askel eteenpäin (Kuva 3); vanhempiin teknologioihin verrattaessa 5G mahdollistaa esimerkiksi paljon korkeamman datan siirtonopeuden, pienemmän latenssin sekä suuremman samanaikaisen käyttäjämäärän (Qualcomm, 2020).

5G-verkon taajuuskaista on jaettu kolmeen osaan, millimetriaallot, keskikaistat ja matalakaistat. 5G-verkon keskikaista oli opinnäytetyön kirjoituksen hetkellä käyttöönotetuin osa 5G-verkkoa. Suuri syy keskikaistan suosioon muihin 5G-verkon kaistoihin verrattaessa oli laitteiston hinta, joka oli lähellä 4G-verkon hintaa. Keskikaistan taajuusalue on 2.4GHz-4.2GHz ja keskikaistalla voidaan teoriassa saavuttaa yli gigabitin sekuntinopeus (GSMA, 2020, s.6).

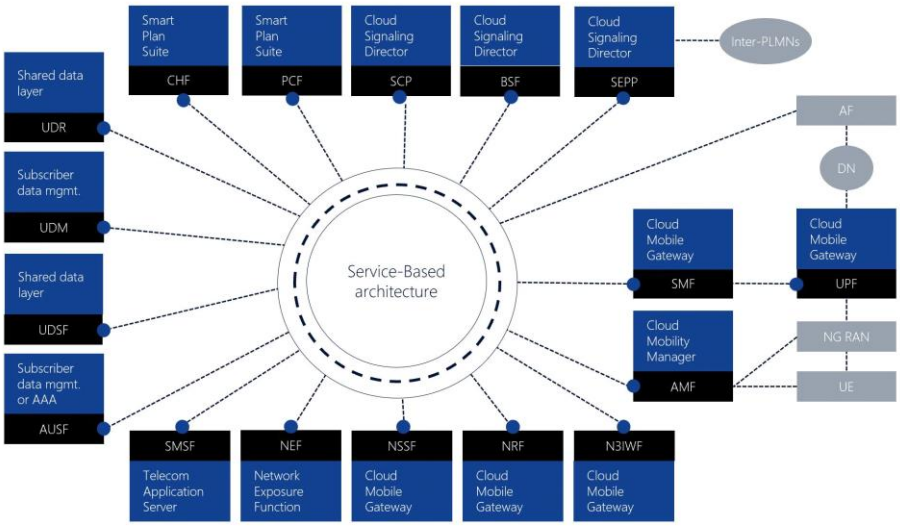
Matalakaista vastaa 4G-verkon nopeuksia, joten tämä ei eroa nykyisistä verkoista juuri mitenkään. Matalakaistan rajoitteena on matalampi alle gigahertsin taajuusalue, joka ei paljon eroa 4G-verkon taajuusalueesta (GSMA, 2020 s.6). Millimetriaallot ovat nopein kaista 1-2 gigabitin latausnopeudella. Taajuus on erittäin korkea verrattuna muihin kaistoihin, mahdollistaen jopa yli 24-73GHz taajuuskaistat. Korkean taajuuskaistan takia signaalin kattavuus ei ole kovin pitkä, vaikka vapaassa ilmatilassa kattavuus ilman suurempaa tiedonsiirtokapasiteetin häviötä onkin noin 2 km (Yong Niu & Yong Li, 2015, s.8). Ongelmaksi muodostuvat rakennukset ja luonnon esteet esimerkiksi puut, joita korkean taajuuskaistan aallot eivät pysty välttämään. Tästä syystä 5G-verkon millimetriaallot vaativat paljon tukiasemia - tästä aiheutuu kyseisten taajuusalueen verkon laajentamisen korkeat kustannukset verrattuna aiempiin 4G-verkkoihin tai 5G-keski- tai matalakaistaverkkoihin.



Kuva 3. "5G is a giant leap" – kuva 5G:n yleisistä hyödyistä verrattuna aiempiin verkkoihin. (Nokia, 2019a)

5G-verkot on suunniteltu korjaamaan nykyisten 4G-verkkojen ongelmia esimerkiksi verkon kaistan ylikuormitus isommissa kaupungeissa. Useissa kaupungeissa verkon kapasiteetti ei enää riitä jakamaan nopeaa yhteyttä käyttäjille, jota tarvitaan esimerkiksi monien nykyisten korkealaatuisten videontoistopalveluiden suoratoistossa. Myös isoihin tapahtumiin 5G on hyvä ratkaisu, koska esimerkiksi suurissa stadioneissa tai muissa tapahtumissa voidaan hyödyntää millimetriaaltoja jakamaan tuhansille käyttäjille samanaikaisesti nopeita yhteyksiä. Tämänkaltaisissa tapahtumissa signaalille ei siis tule samaan tapaan esteitä kuin esimerkiksi kulkiessa kaupungilla. (Holma, Toskala & Nakamura, 2020, ss. 18-21)

Itse 5G-tekniikan ydin (Kuva 4) on palvelupohjainen arkkitehtuuri, joka koostuu monista funktioista.



Kuva 4. 5G Service-Based architecture (Nokia, n.d.h.)

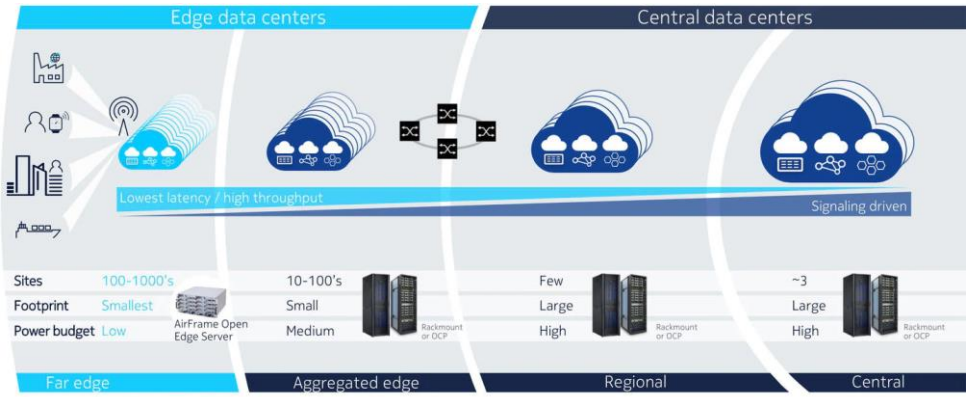
2.3 Edge Cloud

Edge Cloud -palvelimet ovat suunniteltu eliminoimaan loppukäyttäjän latenssiongelmia sekä vähentämään kuormitusta isäntä-pilvipalvelimille. Nämä palvelimet sijaitsevat verkon reunalla ja ovat yleensä maksimissaan noin 30 km etäisyydellä loppukäyttäjästä. Edge Cloud -palvelimia voidaan käyttää esim. CaaS (Container as a Service) (Red Hat, n.d.c) tai IoT (Internet of Things) (Red Hat, n.d.d) palveluita varten. CaaS käytössä kyseinen palvelin mahdollistaisi esimerkiksi kolmannen osapuolen pilvipalveluntarjoajille jaettavia prosessointi ja taltiointi resursseja, joka mahdollistaisi niiden edelleen jakamisen loppukäyttäjälle pienellä viivellä (Ericsson, n.d.a.).

Myös monet palveluntarjoajien määrittämät erityisfunktiot voidaan suorittaa Edge Cloud -palvelimilla. Monet 5G:n toiminnallisuuden kannalta tärkeät funktiot esimerkiksi Central Unit (CU), Radio Unit (RU) ja

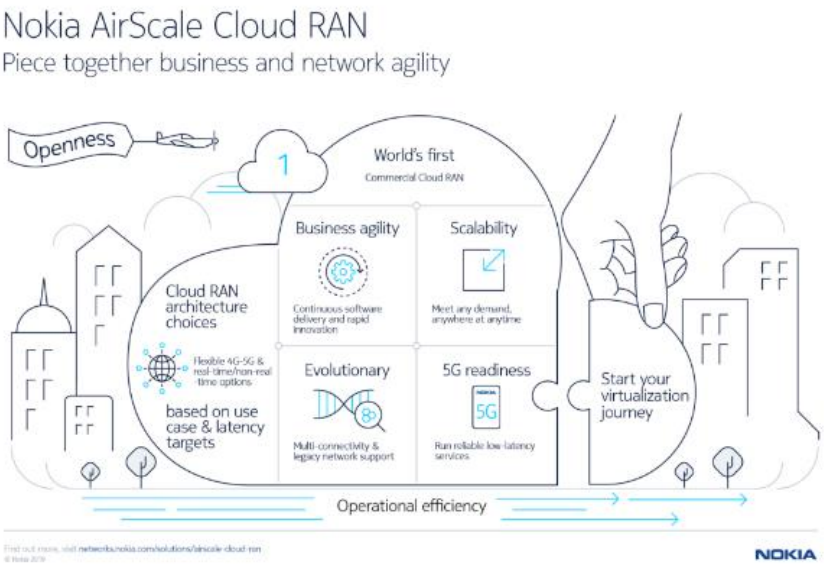
Distributed Unit (DU) (Nokia, 2019b) voidaan suorittaa näillä palvelimilla (Nokia, 2018).

Yksi Nokian tuotteista oli Edge Cloud -palvelut (Kuva 5) (Nokia, n.d.b.) (Nokia, n.d f.) ja laitteisto. Yksi näistä Edge Cloud -palveluihin kuuluvista tuotteista oli AirFrame Open Edge, joka on kolmen rack unitin kokoinen rack mount palvelin, jota Nokia valmisti ja myi asiakkaille. AirFrame Open Edge on suunniteltu matala latenssi ja pieni virrankulutus mielessä, joten Open Edge soveltuu hyvin ML/AI (Machine Learning / Artificial Intelligence) käyttöön. Open Edge on suunniteltu helposti asennettavaksi ja laite on helposti skaalattavissa.



Kuva 5. Yleiskuva Edge Cloudista (Nokia, n.d.c.)

Toinen Edge Cloud portfolioon (Nokia, n.d.f.) kuuluvista tuotteista oli Nokia AirScale Cloud RAN (Kuva 6). AirScale Cloud RAN mahdollistaa radiofunktioiden virtualisoinnin ja suorittamisen verkon edgessä, joka mahdollistaa matalan latenssin ja nopean siirtonopeuden. AirScale Cloud RAN on suunnattu toimimaan myös 5G-laitteiston rinnalla.



Kuva 6. Yleiskuva AirScale Cloud RAN:sta(Nokia, n.d.i)

3 EDGE CLOUD AUTOMAATIO

Tämän opinnäytetyön tehtävänä oli tutkia eri workflow-työkaluja ja selvittää miten kukin workflow-työkalu soveltuu Edge Cloud -automaatioon. Työssä käytettiin soveltuvuuden vertailuun Nokian sisällä laadittuja workflow-työkalujen vaatimuksia ja näiden perusteella verrattiin jokaista työkalua ominaisuuskohtaisesti. Työtä varten työkaluille laadittiin myös pienimuotoinen performanssitesti, josta näkee työkalujen performanssin yksittäisen laitteen ympäristössä. Näissä testeissä käytetyt workflow'it löytyvät liitteenä työn lopusta.

3.1 Automaation hyödyt

Opinnäytetyön kirjoituksen hetkellä Nokia ei tarjonnut Edge Cloud -tuotteissa data centerien infrastruktuurin eli infran asennuksen automaatiota ja tämä automaatio olisi hyödyllinen lisä tuotteiston rinnalla. Isoille asiakkaille voidaan myydä satoja - ellei tuhansia palvelimia, joiden asentaminen oli työn kirjoituksen hetkellä aikaa vievä prosessi. Automaatiolla saataisiin ratkaistua pahimmat ongelmakohdat infran asennuksessa. Lopullinen automaatio keskittyy siis pääosassa infraan, sen asennuksen sekä konfiguraation ongelmiin.

Myös yrityksen sisäisessä työssä ja tutkimuksissa voitaisiin hyödyntää näitä samoja automaatiotyökaluja, kunhan työkaluilla tehty automaatio on alusta lähtien suunniteltu modulaariseksi ja helposti jatkettavaksi.

Ilman automaatioratkaisua infran asennuksessa voi mennä viikkoja, tämä on asiakkaalle paljon turhaan kulutettua aikaa, jota voisi käyttää muihin tehtäviin. Tietenkin myös yrityksen sisäinen toiminta parantuisi, koska infran asennukseen ja konfigurointiin kulutettu aika voitaisiin käyttää itse tuotekehitykseen. Tavoite on ”end-to-end”-tyyppinen automaatio, jossa itse automaatio toteutettaisiin verkon päädyssä ja tämä voitaisiin ratkaista workflow-työkaluilla.

3.2 Toteutukseen käytettävät työkalut

Päädyimme lopputulokseen, että automaatio on hyvä toteuttaa workflow -työkalulla, koska workflow on helpoin asiakkaan sekä yrityksen näkökulmasta toteuttaa ja ylläpitää. Automaatioon ei tarvitsisi muuta kuin itse työkalun ja valmiiksi määritetyt mallit, joita kutsutaan workflow-pohjassa.

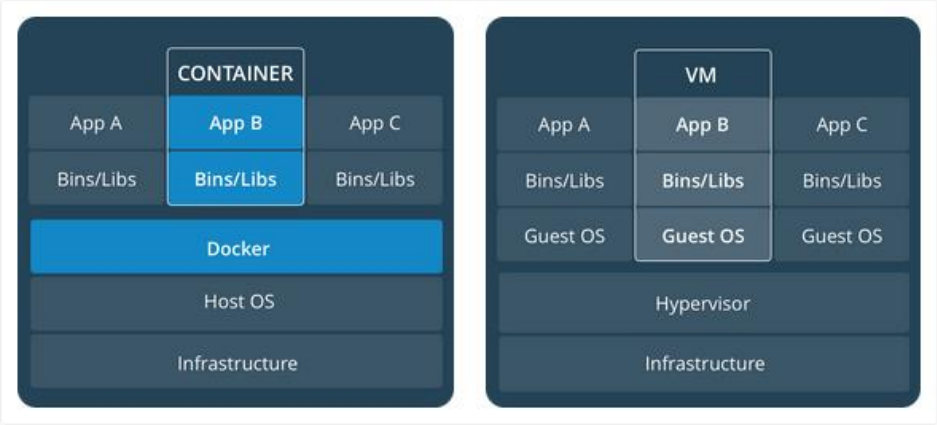
Workflow tulee määritellä asiakkaalle valmiiksi ja määritettyyn workflow'iin asetettaisiin kohdat, joihin voidaan tuoda omia workflow-malleja, joilla voidaan esimerkiksi muokata omaa käyttöjärjestelmää, kustomoida infran asennusta tai tehdä muita jatkotoimenpiteitä. Asennuksen jälkeen asiakas pystyisi automatisoimaan jatkotoimenpiteitä, joihin voisi kuulua verkon konfigurointi, käyttäjien luominen yms.

Workflow-työkaluilla voidaan luoda automaatio myös data centerien raudanhallintaa ja NFVI-tyyppistä infrastruktuuria varten, jossa verkon funktiot virtualisoidaan moniin osiin (Ericsson, n.d.b.). Näissä tapauksissa esimerkiksi REST API-ohjelmointirajapintaa käyttäen voitaisiin lisätä laitteet hallintajärjestelmiin ja näitä järjestelmiä käyttäen ajaa päivitykset laitteistolle, tarkkailla laitteiston resursseja sekä hallita suuria määriä laitteita.

4 SÄILIÖINTI

Container eli säiliö on käyttöjärjestelmätason virtualisointi-paketti, jotka mahdollistaa ohjelmiston ajamisen eristyksessä muusta järjestelmästä. Säiliöitä voidaan ajaa monenlaisilla container engineillä ja näistä tunnetuimpia ovat Docker, CRI-O ja Containerd (Wikipedia, 2020d).

Yksi suurista säiliöiden hyödyistä on ohjelmiston ajaminen mahdollisimman lähellä isäntäkäyttöjärjestelmää, mutta silti eristyksissä muusta käyttöjärjestelmästä (Kuva 7). Tämä toiminnallisuus eroaa vanhoista virtuaalikoneista, jotka virtualisoivat koko tietokoneen toiminnan ja tämän takia virtuaalikoneelle tulee asentaa kaikki ohjelmisto jne. samaan tapaan kuin oikeassakin tietokoneessa. Virtuaalikoneet ovat myös todella raskaita verrattaessa säiliöihin, koska säilöissä ei tarvitse virtualisoida koko käyttöjärjestelmää uudestaan (Docker, 2020).



Kuva 7. Yleiskuva Containerista ja virtuaalikoneesta (Docker, 2020)

Säiliöt suoritetaan isäntäkäyttöjärjestelmän Linux-kernelin päällä käyttäen Linux-kernelin erilaisia resurssien eristysrajapintoja. Näistä yksi on cgroups-rajapinta, joka mahdollistaa prosessien organisoinnin ryhmiin joiden avulla voidaan hallita ja monitoroida säiliöiden saatavilla olevia resursseja (man7.org, 2019a). Kernel namespacet eli nimiavaruudet mahdollistavat laitteiston resurssien jakamisen omiin nimiavaruuksiin. Tämä on tärkeä rajapinta säiliöiden suorittamista varten, koska tällä saavutetaan globaali resurssien käyttö omassa nimiavaruudessa, jonka

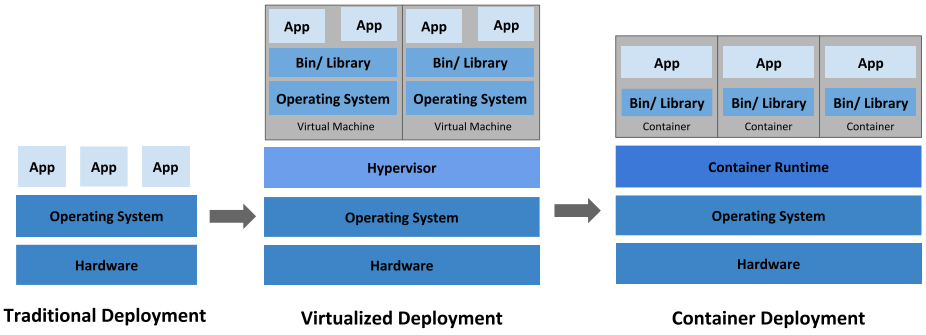
sisällöt näkyvät vain saman nimiavaruuden prosesseille (man7.org, 2019b).

5 KUBERNETES

Kubernetes on säiliöiden orkestraatiojärjestelmä, jolla voidaan tehdä automaatio erilaisten sovellusten asentamista, hallintaa ja suorittamista varten. Kubernetes oli alun perin Googlen suunnittelema, mutta nykyään projektin hallinnasta vastaa Cloud Native Computing Foundation (Cloud Native Computing Foundation, 2020), joka on osa Linux Foundationia.

Kubernetes mahdollistaa ohjelmiston ajamisen monessa eri säiliössä, joka taas mahdollistaa ohjelmiston jakamisen moneen eri osaan. Esimerkiksi verkkopalvelun eri toiminnat esimerkiksi palvelin, tietokanta, ohjelmakoodi ja käyttöliittymä voidaan jakaa eri säiliöihin. Kubernetes pystyy myös hallitsemaan kokonaista ohjelmistopakettia yhtenä kokonaisuutena. Tämän hyötyjä ovat esimerkiksi helppo ohjelmistopaketin päivitys, yhteinen virtuaalilähiverkko palvelujen toiminnan välillä, korkea saatavuus ja ohjelmiston skaalautuvuus.

Kuvasta (Kuva 8) näkee säilöillä toteutetun ohjelmiston hyödyt verrattuna aiempiin toteutuksiin. Säiliöillä saadaan tuotua ohjelmisto tai palvelu mahdollisimman lähelle, koska ei tarvitse erikseen asentaa ja ylläpitää virtuaalikonetta, jotta pystyy suorittamaan ohjelmistoa.



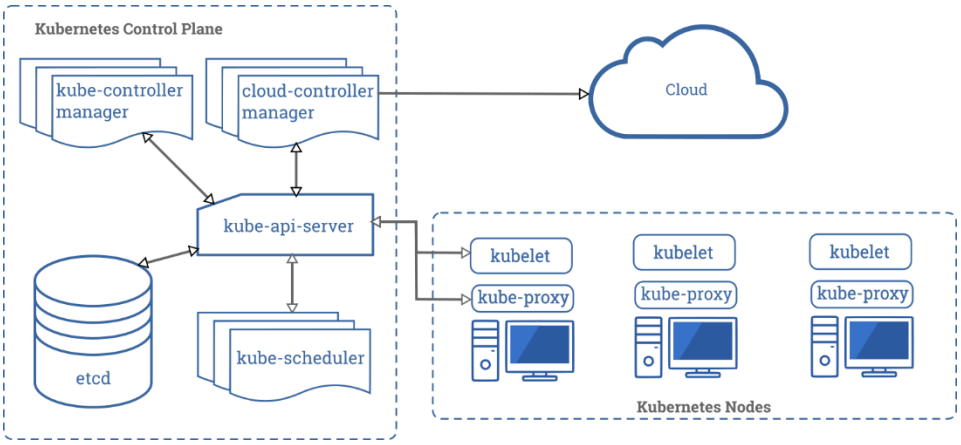
Kuva 8. ”Container Evolution” – yleiskuva säiliöistä verrattuna aiempiin toteutuksiin (Kubernetes, 2020b).

Kubernetes mahdollistaa myös HA-yhteensopivan (High Availability – korkea saatavuus) klusterin konfiguroinnin, joka mahdollistaa Kubernetes komponenttien monistamisen. Tällä voidaan saavuttaa mahdollisimman korkea saatavuus klusterin komponenteille, sekä ajettaville sovelluksille. HA-yhteensopiva klusteri toteutetaan yleensä monistamalla master node (klusteria hallinnoiva päänoodi), joka herää automaattisesti kun käytössä oleva master node kaatuu tai ei ole enää saatavilla (Kubernetes, 2020d).

High Availability on pakollinen ominaisuus tuotannossa, koska koko klusteri hajoaa, jos master node hajoaa.

5.1 Kubernetes klusteri

Kubernetes klusteri koostuu nodeista, joilla ajetaan podeja, jotka sisältävät säiliöitä. Klusterin toiminta on jaettu erilaisiin Control Plane ja Node -komponentteihin, jotka hallitsevat klusterin eri ominaisuuksia. Control Plane -komponentteihin sisältyy kube-apiserver, etcd, kube-scheduler, kube-controller-manager ja cloud-controller-manager (Kubernetes, 2020c).



Kuva 9. “Diagram of a Kubernetes cluster with all the components tied together” – (Kubernetes, 2020c).

Kube-apiserverin tehtävä on jakaa Kubernetesin API-rajapinta ylläpitäjille. Komponentti validoi ja konfiguroi Kubernetes resurssiobjekteja (Pod, Service, ConfigMap yms.) koko klusterille, jotta näitä resursseja pystyy luomaan ja hallitsemaan API-rajapintaa pitkin (Kubernetes, 2020d). Etcd on key-value -tietokanta, johon talletetaan kaikki Kubernetes klusterin tiladata ja muu tarvittava tieto, jotta klusteri pysyy nodejen välillä ajan tasalla (etcd, 2020). Kube-scheduler tarkkailee klusterin nodejen tietoja sekä resursseja ja tämän perusteella varaa uudet podit Kubernetes klusterin nodeille automaattisesti (Kubernetes, 2020e).

Kube-controller-manager komponentti ohjaa controller prosesseja, jotka ovat klusterin eri ohjausyksikköjä. Yksi näistä ohjausyksiköistä on Node Controller, jonka tehtävänä on ohjata klusterin nodeja eri tapahtumien perusteella (Kubernetes, 2020f). Cloud-controller-manager on myös controller prosesseja ohjaava komponentti, mutta klusterin komponenttien ohjauksen sijaan tämä komponentti mahdollistaa kolmannen osapuolen pilvipalveluntarjoajien koodin ajamisen. Tämä komponentti siis mahdollistaa kommunikoinnin klusteriin myös kolmannen osapuolen pilvipalveluista. (Kuberntetes, 2020g)

Node komponentteihin sisältyy kubelet, kubeproxy ja Container Runtime. Nämä komponentit nimensä mukaan ohjaavat klusterin nodeja. Kubelet

on yksi Kubernetesin tärkeimpiä komponentteja, koska tämän tehtävänä on suorittaa ja hallinnoida podeja sekä niiden säiliöitä. Kubelet toimii PodSpec YAML-konfiguraatiotiedostojen määritelmien perusteella, joka luodaan kaikille suoritettaville podeille (Kubernetes, 2020h). Kube-proxy hallinnoi kubernetes klusterin nodejen verkon sääntöjä ja näiden sääntöjen perusteella mahdollistaa podien yhteydet klusterin sisäverkossa tai jopa ulkoverkossa. (Kubernetes, 2020i) Container Runtime on itse säiliöiden ajamisen moottori. Kubernetes tukee erilaisia container runtimeja esim. Docker, containerd ja CRI-O (Kubernetes, 2020j).

5.2 MicroK8s

MicroK8s on kevyt Kubernetes-paketti suunnattu kevyitä ympäristöjä, kuten työpöytäjärjestelmiä varten. Tämän opinnäytetyön työkalujen testaus tehtiin ympäristössä, jossa MicroK8s-paketti käytössä. MicroK8s-paketin mukana tulee Kubernetes komponenttien lisäksi paljon laajennuksia Kubernetesin toiminnan helpottamiseksi. Esimerkiksi työn myöhemmässä vaiheessa performanssitesteissä käytettiin MicroK8s Storage-laajennusta, joka mahdollisti automaattisen taltiointiresurssien määrittämisen (MicroK8s, 2020).

MicroK8s on myös helpompi asentaa laitteistolle verrattaessa perinteiseen Kubernetes ympäristön asennukseen. Linux-ympäristöissä paketti vaatii vain Snap-paketinhallintatyökalun ja itse asennus suoritetaan ”*$ sudo snap install microk8s --classic*”-komennolla.

6 WORKFLOW

Workflow-työkalut ovat monivaiheisten prosessien ja ohjelmakoodien suorittamista varten suunniteltuja työkaluja. Nämä työkalut mahdollistavat erilaisten vaiheiden suorittamisen samanaikaisesti ja riippuvuuspohjaisesti. Yleensä näitä työkaluja käytetään CI/CD-pipelineissä, joissa automatisoidaan ohjelmiston testaus ja kääntäminen (Red Hat n.d.a), mutta niitä voidaan käyttää myös muihin käyttötarkoituksiin esimerkiksi infrastruktuurin ylläpitämisen automaatioon tai ML/AI-prosessointiin (Google, 2020).

6.1 Workflow-työkalun vaatimukset

Työtä varten asetimme vaatimukset, jotka määrittävät mitä ominaisuuksia Workflow-työkalut tarvitsevat, jotta automaatio toimii mahdollisimman hyvin. Ilman vaatimuksia työkalujen selvittäminen tiettyyn käyttötarkoitukseen muuttuu mahdottomaksi prosessiksi, koska kaikilla työkaluilla on omat hyötynsä ja haittansa.

Workflow koostuu useista eri vaiheista. Jokainen vaihe pitää olla mahdollinen käynnistää manuaalisesti tai etänä eri lähteestä käyttäen esimerkiksi REST API-ohjelmointirajapintaa.

Workflow-työkalu olisi hyvä olla Kubernetes natiivi (Red Hat n.d. b), jotta resurssienhallinta olisi helppoa muiden mahdollisten tuotteiden rinnalla. Kubernetes natiivisuuden myötä saamme myös muita hyötyjä esimerkiksi ohjelmointikielestä riippumattomuus, ajonaikaisia yhteensopivuusongelmien eliminointi, versionhallinta työkalujen sekä taltioiden käyttäminen sisäänrakennettujen työkalujen avulla.

Workflow-työkalulla pitää olla mahdollista tallettaa vaiheiden suorituksen aikaisia lokeja jonkinlaiseen kolmannen osapuolen objektisäilöpalveluun. Objektisäilön hyötynä on helppo tietojen haku tietokannasta ja tietokannan nopeus verrattuna tavalliseen paikalliseen taltiointiin. Objektisäilöä tulee olla mahdollista suorittaa lokaalisti tai hyödyntää käyttäen kolmannen osapuolen palveluita, kuten Amazon AWS S3 (Amazon n.d.) tai Google Cloud Storage (Google n.d.).

Workflow-työkalulla täytyy pystyä ajamaan satoja workflow’eja ilman saatavuusongelmia. Workflow-työkalun on tarkoitus myös mahdollistaa isojen data centerien automaatio asiakkaille. Näin ollen työkalun tulee pystyä hallitsemaan resursseja sekä toimia ilman mitään saatavuusongelmia. Työkalun pitää tukea myös resurssien ja samanaikaisten tehtävien yleistä rajoittamista.

Työkalun täytyy sisältää vaiheiden tilanhallinta-työkaluja, jotka esimerkiksi mahdollistavat vaiheen uudelleen käynnistämisen tai koko workflow’n uudelleen suorittamisen. Tilojen pitää pysyä tallessa vaikka palvelin, jossa

työkalua ajetaan käynnistettäisiin uudelleen. Työkalulla pitää pystyä ajamaan töitä silmukassa ja pitää olla mahdollisuus ajaa vaiheita tietyillä ehtovaatimuksilla. Työkalun tärkeä ominaisuus on vaiheiden riippuvuuksien hallinta, koska monet vaiheet automaatiossa vaativat, että aiemmat vaiheet ovat suoritettu ennen uutta vaihetta. Vaiheet tulee pystyä aikakatkaisemaan ongelmakohdissa.

Vaiheiden välisten parametrien ja lokien siirtäminen toiselle vaiheelle on pakollinen ominaisuus. Ilman tätä vaiheet eivät saa tietoa toisiltaan ajon onnistuneisuudesta tai mitä edellinen vaihe palautti. Vaihe pitää pystyä ajoittamaan tiettyjen aikaparametrien mukaan.

Työkalun täytyy olla osa avoimen lähdekoodin projektia, jotta työkalua voidaan käyttää osana tuotetta. Jatkokehityksen kannalta työkalun lisenssi tulisi olla mahdollisimman vapaa kehittää, jotta tulevaisuudessa voidaan kustomoida työkalun toimintaa tai lisätä uusia ominaisuuksia työkaluun. Työkalu tarvitsee myös yksinkertaisen käyttöliittymän workflow'en hallintaa ja tarkkailua varten.

6.2 Argo

6.2.1 Tietoa työkalusta

Argo Workflows on avoimen lähdekoodin Kubernetes natiivi workflow engine, jolla ajetaan workfloweja, jotka koostuvat useista vaiheista. Jokainen vaihe workflowissa on säiliö, jonka sisällä voidaan ajaa miltei mitä tahansa ohjelmakoodia riippumatta ohjelmointikielestä tai muista ympäristön riippuvuuksista. Argon suorituksen aikaisen datan pystyy tallettamaan erilaisiin relaatiotietokantoihin, kuten MySQL ja Postgres. Yksi tärkeä ominaisuus Argossa on tietojen hakeminen ja tallettaminen Artifact Repository-objektisäilöjä (esim. AWS S3, GCS ja MinIO) käyttäen. (GitHub, 2020d)

Intuit (ent. Applatix) perusti Argoproj-projektin, koska yrityksen töissä tarvittiin skaalautuvaa workflow-työkalua, mutta tämän kaltaista työkalua ei ollut saatavilla. Myöhemmin projekti avattiin kaikkien muokattavaksi, kuitenkin pitäen alkuperäiset kehittäjät projektin kehityksen parissa. (Intuit, 2019)

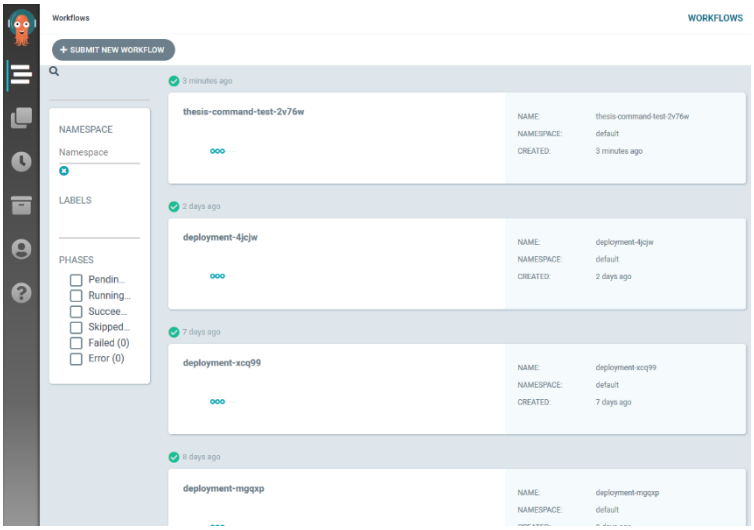
Työkalun Kubernetes natiivisuuden hyödyt ovat runsaat, kaikkien säiliöiden resurssejen hallinta, verkkojen hallinnointi, suorituksenaikaiset lokit, etäajaminen Argon sisäänrakennetun REST API-rajapinnan avulla ja paljon muuta. (GitHub, 2020j)

Argolla voidaan ajaa vaiheita DAG-verkkona (GitHub, 2020d), joka mahdollistaa riippuvuuspohjaisen ajamisen vaiheille. Esimerkiksi yhdellä vaiheella voi olla monta riippuvuutta edellisiin vaiheisiin, joka tarkoittaa

sitä, että tätä kyseistä vaihetta ei voida suorittaa ennen riippuvuuksiksi asetettujen vaiheiden suorittamista.

Ohjelmiston mukana tulee myös Argo CLI, jolla voidaan käsitellä sekä suorittaa workfloweja, tarkastella niiden vaiheita ja itse säiliöitä, joilla vaiheet suoritetaan. (GitHub, 2020q) Paketissa tulee mukana myös Argo Server -komponentti, joka mahdollistaa workflowien monitoroinnin ja käsittelyn Argo UI -selainkäyttöliittymässä (Kuva 10). (GitHub, 2020r) Argo UI on käyttöliittymä, josta voidaan käynnistää workfloweja ja tarkkailla erilaisia workfloweihin liittyviä resursseja. (GitHub, 2020c) Kirjoituksen hetkellä käyttöliittymällä ei ollut varsinaisesti muuta hyödyllistä käyttötarkoitusta, kun nopeasti saatava workflowien tarkkailu. Käyttöliittymää kuitenkin päivitettiin monesti jopa työn kirjoituksen aikana, joten tulevaisuudessa tämä tulee olemaan varmasti parempi.

Työkalussa on myös Kubernetes ”Bearer Token”-pohjainen käyttäjänhallinta, mutta tämä ominaisuus on hetki sitten tuotu työkaluun, joten tämä ei vielä ole kovin pitkälle kehitetty. Käyttäjänhallinnan voi konfiguroida päälle ”*argo server*”-komennolla käyttämällä ”*--auth-mode client|server|hybrid*”-argumenttia. (GitHub, 2020r)



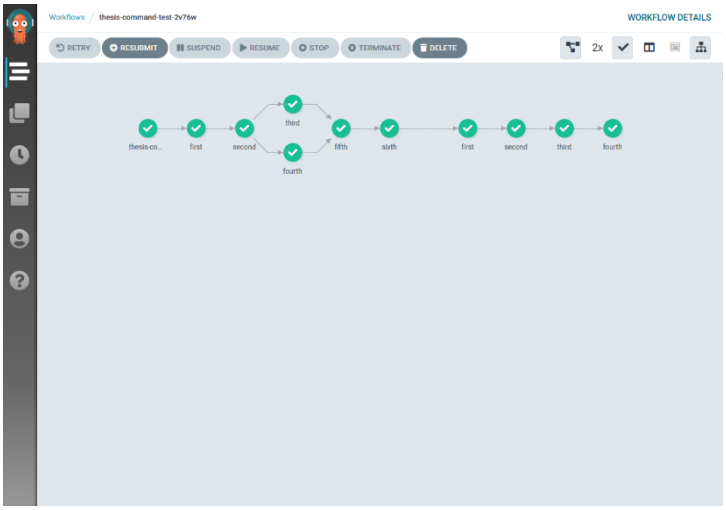
Kuva 10. Argo UI käyttöliittymä selainikkunassa

Argon asennuksen YAML-resurssitiedostossa määritellään erilaisia Kubernetes Role- ja Resource-määritelmiä, joilla mahdollistetaan Kubernetes podien luominen, tarkkailu, suoritus yms. joita Argo tarvitsee toimiaakseen. Argo toimii kolmella säiliöimagella nimiltään *argocli*, *argoexec* ja workflow-controller. Argocli (Docker Hub, 2020a) vastaa käyttöliittymän jakelemisesta ja tämän imagen porttia ja muita resursseja pystyy säätämään Argo CLI -työkalulla. Argoexec (Docker Hub, 2020b) on itse workflowin vaiheiden suoritukseen käytetty image; tällä luodaan säiliöt, joilla suoritetaan workflowien vaiheiden sisällöt. Workflow-controller image (Docker Hub, 2020c) vastaa resurssien jakamisesta ja tämän ConfigMap Kubernetes resurssissa pystyy määritellä suoritettavien workflowien globaaleja muuttujia, esimerkiksi MySQL tai vastaavan

tietokannan työkalulle näkyvää verkkoporttia ja kirjautumistietoja pystyy muokkaamaan tämän resurssin kautta. Myös workflowien toimintaan pystyy vaikuttamaan esimerkiksi *parallelism*-tagilla, jolla saadaan rajoitettua workflowien samanaikaisten suoritusten määrää. (GitHub, 2020k)

6.2.2 Workflow

Argon workflowit (Kuva 11) koostuvat yhdestä workflowin sisältävästä YAML-tiedostosta ja mahdolliseen parametrien tuontiin liittyvästä YAML-tiedostosta. Yksittäisen parametrin voi tuoda erikseen workflowin käynnistyksen yhteydessä, mutta useiden parametrien kohdalla erillinen tiedosto on pakollinen. Workflowien DAG-graafien suunnittelu on helppoa. Riippuvuudet voidaan määrittää ”dependencies”-tagilla vaihekohtaisesti listamuuttujaan YAML-tiedostossa ja yksittäisellä vaiheella voi olla monta riippuvuutta lisättynä tässä listassa. (GitHub, 2020d)



Kuva 11. Argo Workflow Argo UI:ssa

Workflowissa voi olla monia DAG-graafeja, joita voidaan ketjuttaa käynnistymään toisesta DAG:sta. Argon yksi hyöty on workflow templatet, joilla voidaan määrittää workflow malleja. Näitä malleja pystyy kutsumaan ajettavassa workflowissa ja näin ollen on mahdollisuus vaikuttaa itse workflowin sisällön luettavuuteen. (GitHub, 2020l)

Workflow templatet ovat samaan tapaan workfloweja, kuten normaalit workflowit ovat - workflow templaten sisällä voidaan siis ajaa DAG-graafeja ja muita workflowin olennaisia toimintoja. Workflow templateja ei kuitenkaan voida suorittaa samaan tapaan kuin tavallisia workfloweja, vaan näiden tarkoitus on ainoastaan suoraviivaistaa suoritettavia workfloweja.

Argon workflowissa määritetään erikseen säiliöiden parametrit, fyysiset taltiointitilat, ulkopuoliset versionhallinnasta (Git, Gerrit) tai

objektisäiöistä (AWS S3, MinIO) haettavat sisällöt, itse ajettava säiliö, resurssirajoitukset ja säiliöllä ajettava komento. Workflowille pystyy määrittämään myös koko workflowin globaalit parametrit ja taltiointitilat. (GitHub, 2020d)

Argon yksi ominaisuus on Cron workflowit, jotka ovat workfloweja, joiden ainoa ero normaalin workflowin kanssa on Unix-laitteista tunnetun cron-schedulerin hyödyntäminen workflowin ajoittaista suorittamista varten. (GitHub, 2020m) Workflow Archive on valinnainen ominaisuus Argossa, joka on tarkoitettu tietokantaan talletetun suorituksen aikaisen datan hallintaa varten. Yleensä tänne taltioidaan workflowit, joita halutaan säilyttää pidempään. (GitHub, 2020n)

6.3 Apache Airflow

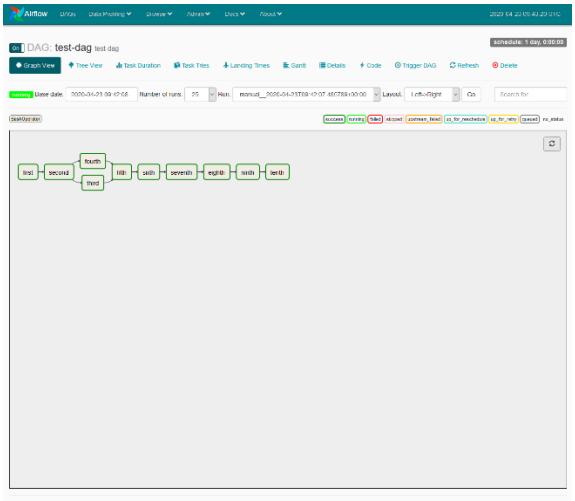
6.3.1 Tietoa työkalusta

Airflow on Apachen kehittämä workflow-työkalu, jolla voidaan suorittaa ohjelmakoodia workflowien eri vaiheissa. (Apache, 2020a) Työkalulla voidaan suorittaa monimutkaisia workfloweja ja vaiheiden suoritus ei vaadi juuri mitään ylimääräisiä resursseja. Airflow on workflow työkaluista tunnetuin, työkalun kehitys alkoi vuonna 2015 ja työn kirjoituksen aikana Git-projektiin oli tehty 8374 commit-päivitystä ja työkalusta oli julkaistu 132 versiota. (GitHub, 2020i) Projektia kehitettiin opinnäytetyön kirjoituksen hetkellä aktiivisesti ja työkalu oli pisimmälle kehitetty tähän työhön valituista työkaluista.

Työkalun voi asentaa SQLite-relaatiotietokannalla, mutta suositeltavaa on asentaa MySQL- tai Postgres-relaatiotietokanta, koska SQLiten kanssa Airflow tukee vain SequentialExecutor-suoritusmoottoria, joka tukee suorittamista vain yhdellä säikeellä. Suoritukseen pystyy myös valita erilaisen suoritusmoottorin esimerkiksi LocalExecutor, KubernetesExecutor tai CeleryExecutor. LocalExecutor ja SequentialExecutor ovat tarkoitettu lokaaliin ajamiseen, mutta LocalExecutor tukee myös monisäikeistä tehtävien suorittamista. KubernetesExecutor ja CeleryExecutor on suunnattu monen noden klusterityylisiä ratkaisuja varten. (Apache, 2019c)

Airflow tukee myös Jinja-templateja, jotka ovat HTML ja XML-pohjia. Nämä pohjat luodaan Python-ohjelmakoodissa ja nämä mahdollistavat datan tuomisen verkkosivupohjaan, jotta kehittäjä ei joudu tuomaan kaikkea dataa manuaalisesti. Tämän työkalun tapauksessa ei tuoda verkkosivupohjaan dataa, vaan tuodaan suoritettavaan koodiin parametri tai makrodataa. Esimerkiksi DAG:in suorituksen yhteydessä voidaan tuoda parametridataa DAG:iin ja näillä pohjilla saadaan kyseinen data ohjelmakoodiin. (GitHub, 2020o)

Airflow'ssa kaikkia workflowien vaiheita kutsutaan operatoreiksi. Python sekä Bash -ohjelmakoodeille on tehty valmiit operatorit ja valmiita operatoreita on myös moniin muihin käyttötarkoituksiin. (Apache, 2020b) Operatoreita voi myös kustomoida ja tehdä omiin käyttötarkoituksiin. Airflow'n asennuksessa tulee myös UI-käyttöliittymä DAG:en hallintaa ja tarkkailua varten (Kuva 12).



Kuva 12. Apache Airflow UI-käyttöliittymä selainikkunassa

6.3.2 DAG

Airflow DAG:ien tekeminen on näistä työkaluista vaivattominta; dokumentaatio on hyvä ja riittävä, toisin kuin muissa työkaluissa. DAG:it Airflow'ssa ovat Python -ohjelmatiedostoja, joissa Python-koodin sekä Bash-komentojen suorittaminen on mahdollista. Bash-komennoilla voi periaatteessa ajaa mitä vaan, joten tälläkin työkalulla pystyisi luomaan automaation Edge Cloudille.

Workflowit ovat todella nopeita suorittaa, koska nämä ajetaan suoraan palvelimella lokaalisti. Lokaalisti suorittaminen on kuitenkin negatiivinen asia tässä tapauksessa; suurin osa Kubernetes klusterin hyödyistä ei ole saatavilla tai vaikeasti toteutettavissa, koska työkalu ei ole Kubernetes natiivi. Tietenkin poikkeuksena aiemmin kappaleessa mainittu KubernetesExecutor, joka mahdollistaa DAGin vaiheiden ajamisen Kubernetesin nodeilla. KubernetesExecutoria käytetään KubernetesOperator kirjaston KubernetesPodOperatorilla DAG:ssa, tämä operator luo Kubernetes podin, jonka määritykset (image, resurssit yms.) löytyvät operatorin rakenteesta. (Apache n.d.)

Työkalusta löytyy kolmannen osapuolen Docker image (GitHub, 2020h) sekä Helm Chart (Helm, 2020) Kubernetes deploymentia varten, mutta näiden skaalautuvuus ei ole samalla tasolla esimerkiksi Argon tai Tektonin kanssa. Myös näiden kustomointi ja käyttäminen on vaikeampaa, koska näiden asennusten kanssa joutuu käsittelemään monia säilöitä ja

workfloweja pitää siirtää monelle säiliölle, jotta Airflow pystyy näitä käsittelemään.

Työkalun suurin haittapuoli on työkalun scheduler, joka ei ole suunniteltu reaaliaikaiseen suoritukseen. Scheduler suorittaa vaiheita sen mukaan miten resursseja on saatavilla ja silloinkin saattaa mennä kymmeniä sekunteja ennen kuin vaihe aloitetaan, vaikka vaiheessa ei tehtäisi muuta kun tulostettaisi tekstiä. Schedulerin nopeuteen voi vaikuttaa vähän säikeiden määrän kasvatuksella ja tietokannan rakenteella, mutta itse schedulerin sisällä tapahtuvaan hitaaseen toimintaan ei pysty vaikuttamaan. (Airflow, n.d.a)

6.4 Tekton

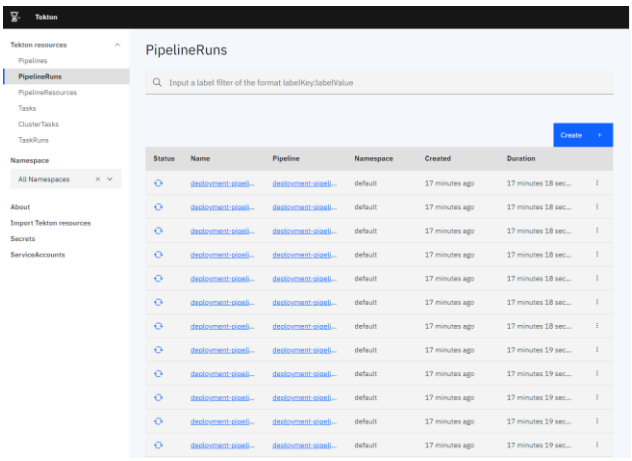
6.4.1 Tietoa työkalusta

Tekton on Googlen kehittämä Kubernetes natiivi CI/CD-pipeline-työkalu, jolla voidaan ajaa monia töitä samanaikaisesti. (Google, n.d.c.) Tekton-projektin yhteistyö partnerit ovat Jenkins X ja GitLab, jotka ovat tunnettuja CI/CD-pipeline ja projektinhallinta työkaluja. Vaikkei Tekton olekaan varsinaisesti workflow-työkalu, on sen vertailu muiden työkalujen kanssa kannattavaa sen Edge Cloud automaatiossa hyödynnettävien ominaisuuksien vuoksi. Suurin osa suunnitelluista vaatimuksista täyttyy tämän työkalun osalta.

Tekton-projekti koostuu monesta eri työkalusta, joita voi ottaa käyttöön valitsemalla tarvittavat työkalut. Tekton ei toimi ilman Pipeline-resurssia, koska Tekton Pipelines koostuu Pipelineistä ja Taskeista, joita ajetaan YAML-tiedoston määritysten mukaan. Vertailussa käytetyt Tekton projektin työkalut olivat Pipeline, CLI ja Dashboard. (GitHub, 2020p)

Tekton CLI mahdollistaa Pipeline-resurssien tarkkailun ja hallinnan komentoriviltä. (GitHub, 2020a) Tekton CLI toimii Mac OSX, Linux ja Windows -käyttöjärjestelmissä, joten työkalun hallinta onnistuu miltei mistä vaan tunnetusta työpöydälle tai palvelimelle suunnatusta käyttöjärjestelmästä.

Tekton Dashboard (Kuva 13) on minimaalinen hallintakäyttöliittymä Tekton-työkalun osia varten. (GitHub, 2020a) Käyttöliittymä on suhteellisen nopea ja suoraviivainen, kunhan ymmärtää suurin piirtein mitä Tektonin erilaiset resurssit ovat. Käyttöliittymästä voidaan käynnistää, poistaa ja suorittaa uudelleen Pipelinejä ja käyttöliittymä mahdollistaa erilaisten suorituksen vaiheiden tarkkailun, sekä vaiheiden logien monitoroinnin. Käyttöliittymästä voidaan myös käynnistää Pipeline hakemalla suoritettava data ulkopuoliselta Git-versionhallintatyökalun palvelimelta.



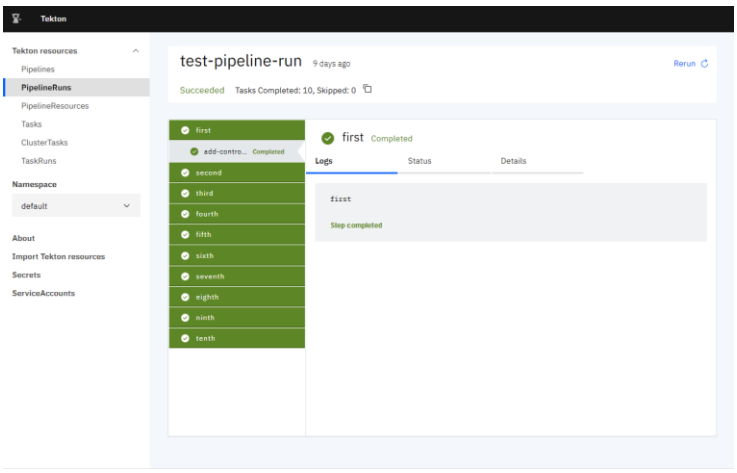
Kuva 13. Tekton Dashboard käyttöliittymä selainikkunassa

Lokaalin taltiointitilan hallintaan Tekton asennus vaatii PVC-resurssien dynaamisen varauksen (Kubernetes, 2020a), jota Kubernetes ei hallitse automaattisesti, joten päädyin käyttämään aiemmassa kappaleessa mainittua MicroK8s-työkalua ja Storage-laajennusta, joka mahdollisti PVC-resurssien dynaamisen varauksen automaattisesti.

Tektonille asennukseen käytettävästä YAML-resurssitiedostosta selviää, että työkalun toiminnan takaavat resurssit ovat samankaltaisia kuin Argon resurssit. (Google, n.d.) Tiedosto sisältää erilaisia Kubernetes CRD, Role, Resource yms. resursseja, joilla mahdollistetaan Tektonin toiminta ja työkalun Kubernetes resurssien manipulointi. Itse suorituksessa käytettävät säiliöimaget ovat jaettu moneen osaan, esimerkiksi Git-versionhallintaan on oma imagensa (Docker Hub, 2019a), käyttäjänhallintaan oma image (Docker Hub, 2019b) yms. Tektonissa on myös controller image (Docker Hub, 2019c) kuten Argossa on workflow-controller ja tämän imagen periaate on sama – image vastaa resurssien jakamisesta podeille. Tektonin Kubernetes ConfigMap resurssit ovat työkalun toiminnan kustomointia varten.

6.4.2 Pipeline

Tekton Pipeline koostuu YAML-tiedostoihin määritellyistä resursseista. Vaadittavia resursseja ovat *Pipeline*, *PipelineRuns*, *Tasks* ja *TaskRuns*. Pipeline on itse ajettava ketju, joka koostuu monista Taskeista. PipelineRuns (Kuva 14) on suoritettu Pipeline ja tämä sisältää kaikki ulkopuoliset sekä muut resurssit, joita koko Pipeline vaatii suorittamista varten.



Kuva 14. Tekton PipelineRun Dashboardissa

Taskit ovat töitä, joita määritetään Pipelineen ajettavaksi. Taskit voivat sisältää monia töitä, mutta näillä töillä ei saada saavutettua riippuvuuksia, tätä varten tarvitaan Pipeline. TaskRun on itse taskin suoritusresurssi ja TaskRun:ssa määritetään tarvittaessa myös suoritettavan taskin resurssit. (GitHub, 2020)

Yksi negatiivinen puoli Tektonissa on Pipelinen luominen. Pipelinen luominen on suhteellisen aikaa vievää ja puuduttavaa, koska jokaiseen vaiheeseen joutuu määrittelemään uudestaan samoja resursseja, joita on jo määritelty aiemmassa vaiheessa. Rivimäärä kasvaa turhan laajaksi ja mahdollinen uudelleen konfigurointi myöhemmässä vaiheessa vaatii moneen tiedostoon muutoksia, joka ei ole ylläpitäjän näkökulmasta suotavaa.

7 VERTAILU

7.1 Ominaisuuksien vertailu

Työkalujen ominaisuuksien vertailussa käytettiin aiemmin määriteltyjä workflow-työkalujen vaatimuksia. Vertailua varten tehdyissä tutkimuksissa selvitettiin työkalujen toiminta käyttämällä ja testaamalla kyseisiä työkaluja. Työkalut ovat todella laajoja, joten ilman vaatimuksia työkalujen toiminnallisuuden vertailu olisi todella hankalaa.

Kaikissa työkaluissa oli REST API-rajapinta, mutta Tektonin kohdalla tämä rajapinta oli implementoitu Kubernetes API-rajapinnan päälle, täten rajapintaa oli hankalampi käyttää. Argossa ja Airflowssa tämä on erikseen toteutettu kokonaisuus ja suunniteltu helppoa käytettävyyttä varten. Airflown REST API-rajapinta on vielä keskeneräinen ominaisuus ja tulee muuttumaan, joten tämän käyttäminen tuotannossa tulee olemaan haasteellista.

Airflow oli ainoa työkalu, joka ei ollut Kubernetes natiivi. Työkalulle on kuitenkin kolmannen osapuolen Docker image sekä Kubernetes deployment, mutta näiden ongelmaksi tuli hankala käytettävyys, puutteellinen tuki muille ohjelmointikielille ja huono tuki näille ratkaisuille. Työkalun säiliöpohjaiset ratkaisut olivat kuitenkin yhtä nopeita kuin lokaaliasennus.

Kaikilla työkaluilla pystyi ajamaan satoja workfloweja ilman suurempia saatavuusongelmia.

Kaikista työkaluista löytyi tuki kolmannen osapuolen objektisäilöpalveluita sekä paikallisia objektisäilöjä varten, mutta työkaluista ainoastaan Argon tuki lokaaleja objektisäilöjä varten oli riittävä lokien tallettamista varten. Airflowlle pystyi myös konfiguroimaan lokaalin objektisäilön, mutta tätä ei pysty hyödyntää, koska tämä tuki on vain PythonOperatorille, joten ei voida suorittaa muuta kuin Python-ohjelmakoodia.

Airflow-työkalulla ei pystynyt rajoittamaan vaiheiden resurssien käyttöä. Argo ja Tekton käyttää Kubernetesin sisäänrakennettua resurssien rajoitusta, joten näillä työkaluilla voi vaikuttaa vaiheiden resursseihin. Tilojen hallinnassa Argo oli työkaluista parhain, koska työkalulla oli kaikki tarvittavat hallintaominaisuudet saatavilla CLI-työkalussa, käyttöliittymässä ja REST API -rajapinnassa. Airflow oli tässä todella kankea, koska hallinta oli mahdollista vain CLI-työkalulla. Tektonissa oli puutteita tämän ominaisuuden suhteen, tilanhallintaa varten oli käytettävissä vain pipelinen uudelleen käynnistäminen ja pysäytys.

Tekton-työkalussa ei ollut silmukoita ja ehtorakenteita vaiheita varten sisäänrakennettuna, nämä joutuu tekemään manuaalisesti ohjelmakoodiin, joka ei ole toivottua toimintaa, koska tätä manuaalisesti Tektonia varten räätälöityä koodia ei voisi enää käyttää suoraa työkalun ulkopuolelle. Ongelmaksi ilmenee myös, että työkalu ei voi luoda uusia haaroja tai vaiheita ehtojen perusteella, joka on suuri ongelma varsinkin jos halutaan tehdä toimintoja dynaamisesti. Esimerkiksi Argossa on mahdollisuus iteroida lista läpi silmukan avulla, samalla luoden uusia vaiheita, jos tämä on tarpeellista.

Kaikissa työkaluissa oli riippuvuuspohjainen suoritus vaiheille. Parametrien siirto vaiheiden välillä oli myös mahdollista kaikilla työkaluilla. Kaikki työkalut olivat osana avoimen lähdekoodin projekteja ja kaikissa työkaluissa oli Apache 2.0 -lisessi, joka mahdollistaa projektin jatkokehityksen tai uuden tuotteen kehittämisen perustuen projektiin, kunhan listataan mitä ominaisuuksia on muokattu (WhiteSource, 2020).

7.2 Performanssivertailussa käytetyt metriikat

Jokaisella työkalulla suoritettiin yksinkertainen tulostustesti, jotta selviäisi itse workflowin käynnistyksessä käytetty aika. Testissä ajettiin Bash-skriptillä simppeli echo-konsolitulostusohjelma.

Työkaluilla suoritettiin myös 10 tehtävää sisältävä workflow, jonka tarkoituksena on simuloida oikeita workfloweja ja täten hahmottaa yleiskuva työkalun performanssista. Taskeissa ajettiin pääosin kevyitä Linux säiliöitä, joilla ajettiin cURL-työkalun komennoilla REST API-rajapinnan GET- ja POST-komentoja ulkoiselle palvelimelle. Nämä suoritetut workflowit liitteinä työn lopussa.

Työkaluilla ajettiin Bash-skriptillä kaikki testit sata kertaa ja komentojen suorituksen nopeutta mitattiin Bashin sisäänrakennetulla time-komennolla.

Työkaluja ajettiin OpenStack (OpenStack, n.d.) pilvipalvelualustalla, jolla voidaan jakaa palvelimen resursseja monille eri käyttäjille. Alustalta varattiin virtuaalikone instanssi, jonka käytettävissä oli 6-ydin 2,5GHz kellotaajudella varustettu prosessori sekä 24G käyttömuistia. Ympäristö ei ole yhtä nopea kuin automaatiossa käytettävä palvelin olisi, joten luvut eivät ole samaa tasoa, kuten isossa Kubernetes klusterissa.

Kubernetes on suunnattu isoja klustereita varten, joten tämänkaltaisissa yhden laitteen ratkaisuissa Kubernetes pohjaiset työkalut ovat kaukana nopeasta, mutta isommassa klusterissa tilanne voisi olla täysin erilainen. Kubernetesin konfiguraatioita ja komponentteja säätämällä klusteria saa paljon nopeammaksi, mutta tässä ympäristössä suurin osa konfiguraatioista pahensi tilannetta, joten näiden säätämisestä ei näissä testeissä ollut hyötyä.

7.3 Performanssivertailujen tulos

Testeissä (Taulukko 1) Airflow oli reippaasti nopein työkalu, seuraavaksi nopein oli Tekton ja viimeisenä Argo. Tilanne muuttui huomattavasti kun Argoon oli konfiguroituna lokaali MinIO-objektisäilö. Airflow ja Tekton eivät tue natiivisti lokaalia objektisäilöä, joten näiden työkalujen kohdalla ei voitu tehdä samanlaista testiskenaariota.

Taulukko 1. Vertailun tulos

Työkalu	Argo	Tekton	AirFlow
100x Bash echo	2m26.127s	2m46.025s	1m43.382s
100x testi workflow	34m2.929s	23m51.274s	14m27.100s
100x testi workflow MinIO-objektisäilöllä	17m34.009s	-	-

Testeistä huomaa myös Tektonin paremman skaalautuvuuden verrattaessa Argoon. Argossa yksinkertaiset tehtävät suoriutuu nopeammin kuin Tektonissa, mutta Tekton on selvästi Argoa parempi monivaiheisessa ajossa. Tämä voi johtua siitä, että Tektonin resurssit ovat jaettu moneen osaan, esimerkiksi Git-versionhallinnan kloonaus tapahtuu omassa tähän käyttötarkoitukseen kustomoidussa säiliössä. Argossa Git-

kloonaus joudutaan tekemään erikseen omassa vaiheessaan tai joka vaiheen alussa uudelleen. Tekton myöskin rajoittaa samanaikaisten resurssien suorituksen määrää automaattisesti, toisin kuin Argo, jossa tämä tehdään manuaalisesti konfiguraatioon.

Airflow'n testaamisessa ongelmaksi syntyi workflow'en rinnakkainen käynnistys, koska Airflow vaatii workflow'n syötteeksi suorituksen käynnistyksen aikamääreen ja työkalu hyväksyy vain yhden workflow'n per sekunti.

Argon toiminnallisuutta pystyy myös konfiguroimaan Kubernetes ConfigMap -konfiguraatioresurssin avulla ja uskon, että tätä säätämällä voisi vielä parantaa Argon performanssia. Tässä ympäristössä suurin osa konfiguraation muutoksista ei tuottanut näkyvää hyötyä, mutta tässä tapauksessa pitää muistaa, että testiympäristö on todella hidas verrattuna tuotannossa käytettävään monen prosessorin ja/tai palvelimen ympäristöön. ConfigMapin sisään implementoitu konfiguraation editointi on kuitenkin suhteellisen uusi ominaisuus Argossa ja vielä ei ole paljon säätövaraa.

7.4 Muita havaintoja

Airflow-työkalu oli nopea, mutta kankea vanhanaikainen lokaaliasennus ei toimi enää moderneissa kehitys- ja tuotantoympäristöissä, joissa yhteensopivuusongelmat tulisi olla minimaaliset tai olemattomat. Vaikka tästä työkalusta oli saatavilla säiliöversio sekä Kubernetes deployment, niin nämä vaihtoehdot eivät vastanneet tarpeita. Plussaa annettava kuitenkin itse workflow'sta, näiden luominen ja ymmärtäminen oli todella nopea prosessi verrattuna muihin työkaluihin.

Tekton on myöskin hyvä työkalu, mutta kuten aiemmin todettiin, tämän tuskainen monivaiheinen Pipelinen kirjoitus oli liian monimutkainen prosessi muihin työkaluihin verrattuna. Pipelinen voi kirjoittaa yhteen YAML-tiedostoon, mutta tämäkin koostuu todellisuudessa monesta YAML-tiedostosta, joille on vain tehty dokumenttieroitin tiedoston perään (merkitään ---).

Argon ja Tektonin välillä ei ole paljon eroa ominaisuuksissa, molemmista löytyy kaikki tarvittavat ominaisuudet jossain muodossa. Tekton on enemmänkin CI/CD Pipeline-työkalu, joten itse suoritettava sisältö ei ole samankaltaista kuin Argossa tai Airflow'ssa. Monihaaraisten vaiheiden seuranta on hankalampaa Tektonissa, koska nämä eivät erotu graafisessa käyttöliittymässä samaan tapaan kuin workflow-työkaluissa.

Argon käyttöliittymä on selvästi todella nopeasti kasaan väännetty viritys, jossa on paljon puutteita, mutta käyttöliittymää on päivitetty työn aikana useasti ja täten parantunut moninkertaisesti paremmaksi muutamassa kuukaudessa. Silti käyttöliittymässä on vielä ongelmia, jos esimerkiksi ajossa on satoja workfloweja. Esimerkiksi myöhemmässä vaiheessa tehdyt

performanssitestit paljastivat performanssiongelmia listanäkymässä; käyttöliittymä ei selvästi tehnyt mitään näytön ulkopuolisten elementtien putsausta lennossa ja piirsi kaikki workflowit kerralla selainikkunaan, täten hidastaen koko selainta – tietyissä tilanteissa jäädyttäen koko selaimen.

Tektonin käyttöliittymästä löytyi myös samankaltaisia ongelmia performanssitestien aikana, mutta piirretyt elementit olivat paljon minimaalisempia, joten käyttöliittymä ei hidastunut mahdottoman hitaaksi. Poikkeuksena TaskRunien tarkkailuvälilehti, jossa listataan kaikkien workflow'en vaiheet, joita on yli tuhat jälkimmäisessä performanssitestissä.

Airflow'n käyttöliittymä oli kaikin tavoin suhteellisen vanhanaikainen ja missään vaiheessa ruudulla ei tapahtunut niin paljon, että selain olisi hidastunut. Käyttöliittymä ei tuntunut olevan mitenkään dynaaminen ja tämä on suuri puute, jos halutaan reaaliaikaisesti monitoroida asennusta päivittämättä sivua manuaalisesti.

Airflow'n käyttöliittymälle kuitenkin annettava plussaa DAG:en monipuolisesta datan esityksestä ja monien erilaisten kaavioiden luomisesta. Esimerkiksi kaikista DAG:n vaiheista löytyy Gantt-kaavioita ja muita graafeja, joista näkee tarkkaan kuinka kauan kussakin vaiheessa meni. Argossa oli myös Gantt-kaavio, mutta tämä on suhteellisen minimaalinen verrattuna Airflow'n kaavioon.

8 YHTEENVETO

Työkalujen vertailun jälkeen saavuin lopputulokseen, että Argo olisi mielestäni paras työkalu tähän käyttötarkoitukseen, koska ominaisuusvertailun perusteella Argosta löytyy suurin osa ominaisuuksista ja työkalun käytettävyys on helppoa. Tekton olisi myös käypä työkalu, mutta työkalun hankala käyttö ja sisäänrakennetun ehtojen sekä silmukoiden puute rajoittavat työkalua huomattavasti. Airflow ei ollut kovin dynaaminen toiminnaltaan ja työkalu ei ole selvästi suunniteltu reaaliaikaiseen ajoon, joten seuraavien vaiheiden suoritus ei tapahtunut heti edellisen vaiheen jälkeen, joka ei ole suotavaa massiivisten data centerien automaatiossa.

Argoon oli mahdollista konfiguroida MinIO-objektisäilö, jolla voitiin parantaa työkalun performanssia. Tässä konfiguraatiossa Argon kaikki workflow vaiheiden lokit ja muu data siirrettäisiin MinIO-objektisäilöön ja täten saavutetaan nopeampi suoritusaika verrattaessa vakiotilaan, jossa Argo hakee lokit ja muut resurssit Kubernetes klusterin etcd-tietokannasta. Airflow'n performanssiin voisi myös vaikuttaa muokkaamalla työkalun Scheduler-komponentin konfiguraatiota vastaamaan ajoympäristöä (suoritusta varten käytettävät säikeet yms.). Airflow'n performanssiin voi myös vaikuttaa muokkaamalla työkalun airflow.cfg-konfiguraatiotiedostoa. (Airflow n.d.b) Argo ja Tekton -työkalujen performanssiin voitiin vaikuttaa näiden Kubernetes ConfigMap -konfiguraatioiden avulla.

8.1 Projektin kulku

Projekti aloitettiin vuoden 2020 alussa ja projektin ensimmäinen vaihe oli tehdä toimiva demo Argo-työkalulla, jotta saataisiin testattua, että soveltuvatko workflow-työkalut Edge Cloud -automaatioon. Yrityksessä ei ollut varsinaista tekijää näiden työkalujen parissa, joten kaikki tutkiminen ja kehittäminen automaatioon liittyen oli vastuullani. Aluksi hankalaa oli hahmottaa miten näitä työkaluja kannattaisi käyttää, jotta saataisiin tuote, jota voitaisiin myydä jatkossa asiakkaille päätuotteen rinnalla.

Loppujenlopuksi kuitenkin saatiin kasaan malli toimivasta Edge Cloud -automaatiosta, josta syystä työ onnistui mielestäni hyvin.

8.2 Opinnäytetyön prosessi

Opinnäytetyön kirjoittamista hankaloitti kirjoittamisen hetkellä ollut COVID19-virusepidemia ja tämän aiheuttama etätyöskentely. Tästä syystä ajatusten vaihtaminen muiden kanssa ei ollut samalla tavalla mahdollista verrattaessa lähityöskentelyyn. Itse työkaluista ja tekniikoista kirjoittaminen oli todella mukavaa itselle mieleisen aiheen takia. Pääsin tutkimaan myös yrityksen historiaa ja tuotteita jossain määrin. Esimerkiksi 5G-verkon toiminnan tutkimisessa opin paljon uutta mobiiliverkkojen

toiminnasta yleisesti ja myös 5G-teknologian ytimen toimintaan sai paremman kuvan.

Opinnäytetyöhön liittyen tein töissä useita demoja Edge Cloud-automaatiosta Argo-työkalulla ja pääsin kehittämään omia esiintymistaitoja sekä stressinsietokykyä ongelmatilanteissa. Myös erilaisia skriptejä ja ohjelmakoodia pääsi työstämään demoja sekä muita opinnäytetyön vertailumetriikoita varten.

Tämä työ sai minut innostumaan myös tekemään muutamia pieniä bugikorjauksia ja parannuksia Argon GitHub-projektiin ja uskon, että myös jatkossa tulen tekemään korjauksia tähän projektiin ja myös muihin avoimen lähdekoodin projekteihin, koska työn haasteiden ansiosta koin, että opin paljon projektien struktuurista ja toteutusten rakenteesta.

LÄHTEET

Amazon (n.d.) Amazon S3. Haettu 16.3.2020 osoitteesta https://aws.amazon.com/s3/

Apache (2020a). Apache Airflow. Haettu 16.3.2020 osoitteesta https://airflow.apache.org/

Apache (2020b). Airflow operators. Haettu 25.3.2020 osoitteesta https://airflow.apache.org/docs/stable/_api/airflow/operators/index.html

Apache (2020c). airflow.executors. Haettu 14.4.2020 osoitteesta https://airflow.apache.org/docs/stable/_api/airflow/executors/index.html

Apache (n.d.) Kubernetes Executor Haettu 14.4.2020 osoitteesta https://airflow.apache.org/docs/1.10.1/kubernetes.html

Airflow (n.d.a) Scheduler. Haettu 11.5.2020 osoitteesta https://airflow.readthedocs.io/en/latest/scheduler.html

Airflow (n.d.b). Faq. Haettu 26.5.2020 osoitteesta https://airflow.apache.org/docs/stable/faq.html#how-can-my-airflow-dag-run-faster

Cloud Native Computing Foundation (2020). Building Sustainable Ecosystems for Cloud Native Software. Haettu 16.3.2020 osoitteesta https://www.cncf.io/

Docker (2020). Orientation and setup. Haettu 6.4.2020 osoitteesta https://docs.docker.com/get-started/

Docker Hub (2020a). Argoproj/argocli. Haettu 13.5.2020 osoitteesta https://hub.docker.com/r/argoproj/argocli

Docker Hub (2020b). Argoproj/argoexec. Haettu 13.5.2020 osoitteesta https://hub.docker.com/r/argoproj/argoexec

Docker Hub (2020c). Argoproj/workflow-controller. Haettu 13.5.2020 osoitteesta https://hub.docker.com/r/argoproj/workflow-controller

Docker Hub (2019a). Tekton/git-init. Haettu 13.5.2020 osoitteesta https://hub.docker.com/r/tekton/git-init-4874978a9786b6625dd8b6ef2a21aa70

Docker Hub (2019b). Tekton/creds-init. Haettu 13.5.2020 osoitteesta

https://hub.docker.com/r/tekton/creds-init-c761f275af7b3d8bea9d50cc6cb0106f

Docker Hub (2019c). Tekton/controller. Haettu 13.5.2020 osoitteesta https://hub.docker.com/r/tekton/controller-10a3e32792f33651396d02b6855a6e36

Ericsson (n.d.a) Cloud Infrastructure. Haettu 23.4.2020 osoitteesta https://www.ericsson.com/en/portfolio/digital-services/cloud-infrastructure

Ericsson (n.d.b). NFVI. Haettu 15.4.2020 osoitteesta https://www.ericsson.com/en/portfolio/digital-services/cloud-infrastructure/nfvi

etcd (2020). The etcd documentation. Haettu 4.5.2020 osoitteesta https://etcd.io/docs/

GitHub (2020a) Tekton. Haettu 16.3.2020 osoitteesta https://github.com/tektoncd/

GitHub (2020b). Argoproj – Get stuff done with Kubernetes. Haettu 16.3.2020 osoitteesta https://github.com/argoproj/argo

GitHub (2020c). Argo Getting Started. Haettu 16.3.2020 osoitteesta https://github.com/argoproj/argo/blob/master/docs/getting-started.md

GitHub (2020d). Argo Workflows: Documentation by Example. Haettu 16.3.2020 osoitteesta https://github.com/argoproj/argo/blob/master/examples/README.md

GitHub (2020f). Installing Tekton Pipelines. Haettu 16.3.2020 osoitteesta https://github.com/tektoncd/pipeline/blob/master/docs/install.md

GitHub (2020g). Tekton Pipelines Tutorial. Haettu 16.3.2020 osoitteesta https://github.com/tektoncd/pipeline/blob/master/docs/tutorial.md

GitHub (2020h). Docker Apache Airflow. Haettu 24.3.2020 osoitteesta https://github.com/puckel/docker-airflow

GitHub (2020i). Awesome-workflow-engines. Haettu 25.3.2020 osoitteesta https://github.com/meirwah/awesome-workflow-engines

GitHub (2020j). REST API. Haettu 11.5.2020 osoitteesta https://github.com/argoproj/argo/blob/master/docs/rest-api.md

GitHub (2020k). Workflow-controller-configmap. Haettu 11.5.2020 osoitteesta

https://github.com/argoproj/argo/blob/master/docs/workflow-controller-configmap.yaml

GitHub (2020l). Workflow Templates. Haettu 11.5.2020 osoitteesta https://github.com/argoproj/argo/blob/master/docs/workflow-templates.md

GitHub (2020m). Cron Workflows. Haettu 11.5.2020 osoitteesta https://github.com/argoproj/argo/blob/master/docs/cron-workflows.md

GitHub (2020n). Workflow Archive. Haettu 11.5.2020 osoitteesta https://github.com/argoproj/argo/blob/master/docs/workflow-archive.md

GitHub (2020o). Templating with Jinja. Haettu 11.5.2020 https://github.com/apache/airflow/blob/master/docs/tutorial.rst#templating-with-jinja

GitHub (2020p). Tekton. Haettu 11.5.2020 https://github.com/tektoncd

GitHub(2020q). Argo CLI. Haettu 13.5.2020 osoitteesta https://github.com/argoproj/argo/blob/master/docs/cli.md

GitHub (2020r). Argo Server. Haettu 13.5.2020 osoitteesta https://github.com/argoproj/argo/blob/master/docs/argo-server.md

GitHub (2020s). Tasks. Haettu 13.5.2020 osoitteesta https://github.com/tektoncd/pipeline/blob/master/docs/tasks.md

GitHub (2020t). Workflow Templates. Haettu 13.5.2020 osoitteesta https://github.com/argoproj/argo/blob/master/docs/workflow-templates.md

Google (2020) Machine learning workflow. Haettu 16.3.2020 osoitteesta https://cloud.google.com/ai-platform/docs/ml-solutions-overview

Google (n.d.a) Cloud Storage. Haettu 16.3.2020 osoitteesta https://cloud.google.com/storage

Google (n.d.b) Tekton Release. Haettu 11.5.2020 https://storage.googleapis.com/tekton-releases/pipeline/latest/release.yaml

Google (n.d.c) Tekton. Haettu 11.5.2020 osoitteesta https://cloud.google.com/tekton

GSMA (2020). 5G Spectrum. Haettu 27.4.2020 osoitteesta https://www.gsma.com/spectrum/wp-content/uploads/2020/03/5G-Spectrum-Positions.pdf

Holma, H., Toskala, A. & Nakamura, T. (2020). *5G Technology 3GPP New Radio.* Chichester: Wiley

Helm (2020). Airflow / Celery. Haettu 24.3.2020 osoitteesta https://hub.helm.sh/charts/stable/airflow

Intuit (2019). Developing Community Open source technology at Intuit. Haettu 14.4.2020 osoitteesta https://opensource.intuit.com/

Kubernetes (2020a). PersistentVolumeClaims. Haettu 16.3.2020 osoitteesta https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims

Kubernetes (2020b). What is Kubernetes. Haettu 16.3.2020 osoitteesta https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes (2020c). Kubernetes Components. Haettu 27.3.2020 osoitteesta https://kubernetes.io/docs/concepts/overview/components/

Kubernetes (2020d). kube-apiserer. Haettu 4.5.2020 osoitteesta https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/

Kubernetes (2020d). Set up High-Availability Kubernetes Masters. Haettu 14.4.2020 osoitteesta https://kubernetes.io/docs/tasks/administer-cluster/highly-available-master/

Kubernetes (2020e). Kube-scheduler. Haettu 4.5.2020 osoitteesta https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/

Kubernetes (2020f). Kube-controller-manager. Haettu 4.5.2020 osoitteesta https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/

Kubernetes (2020g). Cloud Controller Manager. Haettu 4.5.2020 osoitteteesta https://kubernetes.io/docs/concepts/architecture/cloud-controller/

Kubernetes (2020h). Kubelet. Haettu 4.5.2020 https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/

Kubernetes (2020i). Kube-proxy. Haettu 4.5.2020 https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/

Kubernetes (2020j). Container runtimes. Haettu 4.5.2020 https://kubernetes.io/docs/setup/production-environment/container-runtimes/
man7.org (2019a). cgroups – Linux control groups. Haettu 6.4.2020 osoitteesta http://man7.org/linux/man-pages/man7/cgroups.7.html

man7.org (2019b). namespaces – overview of Linux namespaces. Haettu 6.4.2020 osoitteesta http://man7.org/linux/man-pages/man7/namespaces.7.html

MicroK8s (2020). MicroK8s Addons. Haettu 17.3.2020 osoitteesta https://microk8s.io/docs/addons

Mobile phone history (n.d.). Nokia 1011. Haettu 1.4.2020 osoitteesta https://www.mobilephonehistory.co.uk/nokia/nokia_1011.php

Nokia (2019a). 5G is a giant leap. Haettu 13.3.2020 osoitteesta https://www.nokia.com/networks/5g/mobile/

Nokia (2019b). A giant leap for 5G Cloud RAN. Haettu 2.4.2020 osoitteesta https://www.nokia.com/blog/giant-leap-5g-cloud-ran/

Nokia (2018) Nokia AirFrame Cloud Infrastructure for Real-time applications (NCIR). Haettu 12.3.2020 osoitteesta https://onestore.nokia.com/asset/205140

Nokia (2018). The edge cloud. Haettu 2.4.2020 osoitteesta https://onestore.nokia.com/asset/202184

Nokia (n.d.a.) What we do. Haettu 12.3.2020 osoitteesta https://www.nokia.com/fi_fi/tietoa-nokiasta/mita-teemme/#for-service-providers

Nokia (n.d.b.) Edge Cloud. Haettu 12.3.2020 osoitteesta https://www.nokia.com/networks/solutions/edge-cloud/

Nokia (n.d.c.) OpenEdge Architecture. Haettu 12.3.2020 osoitteesta https://www.nokia.com/sites/default/files/styles/scale_1440_no_crop/public/2019-04/openedge_architecture_0_4.jpg

Nokia (n.d.d.) AirFrame open edge server. Haettu 12.3.2020 osoitteesta https://www.nokia.com/networks/products/airframe-open-edge-server/

Nokia (n.d.e.) Deploy extraordinary 5G networks. Haettu 12.3.2020 osoitteesta https://www.nokia.com/networks/5g/mobile/

Nokia (n.d.f.) Edge Cloud portfolio. Haettu 13.3.2020 osoitteesta https://www.nokia.com/networks/portfolio/edge-cloud/

Nokia (n.d.g.) Nokia Bell Labs. Haettu 18.3.2020 osoitteesta https://www.nokia.com/innovation/nokia-bell-labs/

Nokia (n.d.h.) 5G Core. Haettu 2.4.2020 osoitteesta https://www.nokia.com/networks/portfolio/5g-core/#defining-a-new-5g-core

Nokia (n.d.i) AirScale Cloud RAN. Haettu 4.5-2020 osoitteesta https://www.nokia.com/networks/solutions/airscale-cloud-ran/

OpenStack (n.d.). OpenStack. Haettu 15.4.2020 osoitteesta https://www.openstack.org/

Qualcomm (2020). What is 5G. Haettu 27.4.2020 osoitteesta https://www.qualcomm.com/invention/5g/what-is-5g

Red Hat (n.d.a) What is CI/CD ?. Haettu 13.3.2020 osoitteesta https://www.redhat.com/en/topics/devops/what-is-ci-cd

Red Hat (n.d.b) Kubernetes-native infrastructure. Haettu 16.3.2020 osoitteesta https://www.openshift.com/learn/topics/kubernetes-native-infrastructure

Red Hat (n.d.c) What is CaaS. Haettu 2.4.2020 osoitteesta https://www.redhat.com/en/topics/cloud-computing/what-is-caas

Red Hat (n.d.d) What does IoT mean for business? Haettu 2.4.2020 osoitteesta https://www.redhat.com/en/topics/internet-of-things

Yong Niu & Yong Li (2016). A Survey of Millimeter Wave (mmWave) Communications for 5G: Opportunities and Challenges. Haettu 22.4.2020 osoitteesta https://arxiv.org/pdf/1502.07228.pdf

The Verge (2020). Nokia N9 review. Haettu 1.4.2020 osoitteesta https://www.theverge.com/2011/10/22/2506376/nokia-n9-review

WhiteSource (2020). Top 10 Apache License Questions Answered. Haettu 23.4.2020 osoitteesta https://resources.whitesourcesoftware.com/blog-whitesource/top-10-apache-license-questions-answered

Wikipedia (2020a). Nordic Mobile Telephone. Haettu 1.4.2020 osoitteesta https://en.wikipedia.org/wiki/Nordic_Mobile_Telephone

Wikipedia (2020b). MeeGo. Haettu 1.4.2020 osoitteesta https://en.wikipedia.org/wiki/MeeGo

Wikipedia (2020c). Nokia N9. Haettu 27.4.2020 osoitteesta https://en.wikipedia.org/wiki/Nokia_N9

Wikipedia (2020d). OS-level virtualization. Haettu 4.5.2020 osoitteesta https://en.wikipedia.org/wiki/OS-level_virtualization

Liite 1

Argo testi Workflow

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: thesis-command-test-
spec:
  entrypoint: test-dag
  volumes:
  - name: workdir
    persistentVolumeClaim:
      claimName: argo-pvc
  arguments:
    parameters:
    - name: test-parameter
      value: "{{inputs.parameters.test-parameter}}"
  templates:
  - name: test-dag
    dag:
      tasks:
      - name: first
        template: jq
        arguments:
          parameters:
          - name: command
            value: echo "first step, testing parameters {{workflow.parameters.test-parame-
ter}}"

      - name: second
        dependencies: [ first ]
        template: jq
        arguments:
          parameters:
          - name: command
            value: curl -sk -X POST -H "Content-Type:application/json" -d '{"test-data":"Hello
World"}' http://localhost:5000/post-data/

      - name: third
        dependencies: [ second ]
        template: jq
        arguments:
          parameters:
          - name: command
            value:  echo 'third step'

      - name: fourth
        dependencies: [ second ]
        template: jq
```

```
      arguments:
        parameters:
        - name: command
          value: echo 'fourth step'

    - name: fifth
      dependencies: [ third, fourth ]
      template: jq
      arguments:
        parameters:
        - name: command
          value: curl -sk -H "Content-Type:application/json" http://localhost:5000/get-
data/

    - name: sixth
      dependencies: [ fifth ]
      template: jq
      arguments:
        parameters:
        - name: command
          value: echo 'sixth step'

    - name: test-dag-2-exec
      dependencies: [ sixth ]
      template: test-dag-2

 - name: test-dag-2
   dag:
     tasks:
     - name: first
       template: jq
       arguments:
         parameters:
         - name: command
           value: echo 'first step'
     - name: second
       dependencies: [ first ]
       template: jq
       arguments:
         parameters:
         - name: command
           value: curl -sk -X POST -H "Content-Type:application/json" -d '{"test-data":"Hello
World"}' http://localhost:5000/post-data/

     - name: third
       dependencies: [ second ]
       template: jq
       arguments:
         parameters:
```

```
        - name: command
          value: curl -sk -X POST -H "Content-Type:application/json" -d '{"test-data":"Hello
World"}' http://localhost:5000/post-data/
    - name: fourth
      dependencies: [ third ]
      template: jq
      arguments:
        parameters:
        - name: command
          value: curl -sk -H "Content-Type:application/json" http://localhost:5000/get-
data/

 - name: jq
   outputs:
     parameters:
     - name: command-output
       valueFrom:
         path: /tmp/command_output.txt
   inputs:
     parameters:
     - name: command
   container:
     image: tkilpela/ubuntu-jq:latest
     command: [sh, -c]
     args: ["{{inputs.parameters.command}} > /tmp/command_output.txt; cat
/tmp/command_output.txt"]
     volumeMounts:
     - name: workdir
       mountPath: /mnt/vol
     resources:
        limits:
          memory: 100Mi
          cpu: 0.1
```

Liite 2

Tekton testi Pipeline ja muut resurssit

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: first
spec:
  steps:
    - name: first
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
      args:
        - -c
        - echo 'first'
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: second
spec:
  results:
    - name: test-param
      description: test parameter
  steps:
    - name: second
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
      args:
        - -c
        - curl -sk -X POST -H "Content-Type:application/json" -d '{"test-data":"Hello
World"}' http://localhost:5000/post-data/ > /tekton/results/test-param
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: third
spec:
  steps:
    - name: third
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
      args:
        - -c
        - echo 'third'
```

```
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: fourth
spec:
  steps:
    - name: fourth
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
      args:
        - -c
        - echo 'fourth'
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: fifth
spec:
  steps:
    - name: fifth
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
      args:
        - -c
        - curl -s -k -X GET http://localhost:5000/get-data/
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: sixth
spec:
  steps:
    - name: sixth
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
      args:
        - -c
        - echo 'sixth'

---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: seventh
spec:
```

```
  steps:
    - name: seventh
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
      args:
        - -c
        - echo 'seventh'
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: eighth
spec:
  steps:
    - name: eighth
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
      args:
        - -c
        - curl -sk -X POST -H "Content-Type:application/json" -d '{"test-data":"Hello
World"}' http://localhost:5000/post-data/
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: ninth
spec:
  steps:
    - name: ninth
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
      args:
        - -c
        - curl -sk -X POST -H "Content-Type:application/json" -d '{"test-data":"Hello
World"}' http://localhost:5000/post-data/
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: tenth
spec:
  steps:
    - name: tenth
      image: tkilpela/ubuntu-jq:latest
      command:
        - sh
```

```
    args:
      - -c
      - curl -s -k -X GET http://localhost:5000/get-data/
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: test-pipeline
spec:
  tasks:
    - name: first
      taskRef:
        name: first

    - name: second
      taskRef:
        name: second
      runAfter:
        - first

    - name: third
      taskRef:
        name: third
      params:
        - name: test-param
          value: "$(tasks.second.results.test-param)"
      runAfter:
        - second

    - name: fourth
      taskRef:
        name: fourth
      runAfter:
        - third

    - name: fifth
      taskRef:
        name: fifth
      runAfter:
        - fourth

    - name: sixth
      taskRef:
        name: sixth
      runAfter:
        - fifth

    - name: seventh
      taskRef:
```

```
      name: seventh
    runAfter:
      - sixth

  - name: eighth
    taskRef:
      name: eighth
    runAfter:
      - seventh

  - name: ninth
    taskRef:
      name: ninth
    runAfter:
      - eighth

  - name: tenth
    taskRef:
      name: tenth
    runAfter:
      - ninth
---

apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: test-pipeline-run
spec:
  pipelineRef:
    name: test-pipeline
  serviceAccountName: 'default'
```

Liite 3

Airflow testi DAG

```
from datetime import timedelta
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago
from airflow.utils import timezone
from datetime import datetime


default_args = {
    'name': 'test-dag',
    'start_date': str(datetime.today()).split()[0],
    'schedule_interval': None
}

dag = DAG(
    'test-dag',
    default_args=default_args,
    description='test-dag',
    tags=['dag'],
)

t1 = BashOperator(
    task_id='first',
    bash_command='echo "first"',
    dag=dag,
)
t2 = BashOperator(
    task_id='second',
    bash_command="""curl -sk -X POST -H "Content-Type:application/json" -d '{"test-
data":"Hello World"}' http://localhost:5000/post-data/""",
    # xcom_push enables the possibility to push data to other operators in dag
    xcom_push=True,
    dag=dag,
)
t3 = BashOperator(
    task_id='third',
    bash_command="""echo "third, test parameter pulling {{
ti.xcom_pull(task_ids='second') }}" """,
    dag=dag,
)
t4 = BashOperator(
    task_id='fourth',
    bash_command='echo "fourth"',
    dag=dag,
)
t5 = BashOperator(
```

```
    task_id='fifth',
    bash_command="""curl -s -k -X GET http://localhost:5000/get-data/""",
    dag=dag,
)
t6 = BashOperator(
    task_id='sixth',
    bash_command='echo "sixth"',
    dag=dag,
)
t7 = BashOperator(
    task_id='seventh',
    bash_command='echo "seventh"',
    dag=dag,
)
t8 = BashOperator(
    task_id='eighth',
    bash_command="""curl -sk -X POST -H "Content-Type:application/json" -d '{"test-
data":"Hello World"}' http://localhost:5000/post-data/""",
    dag=dag,
)
t9 = BashOperator(
    task_id='ninth',
    bash_command="""curl -sk -X POST -H "Content-Type:application/json" -d '{"test-
data":"Hello World"}' http://localhost:5000/post-data/""",
    dag=dag,
)
t10 = BashOperator(
    task_id='tenth',
    bash_command="""curl -s -k -X GET http://localhost:5000/get-data/""",
    dag=dag,
)

t1 >> t2
t2 >> [t3, t4]
[t3, t4] >> t5
t5 >> t6
t6 >> t7
t7 >> t8
t8 >> t9
t9 >> t10
```